VŠB - Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra Informatiky

On-line nástroj pro konverzi různých datových formátů

On-line Tool for Converting Different Data Structures

2017 David Kolek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student:	**David Kolek**
Studijní program:	B2647 Informační a komunikační technologie
Studijní obor:	2612R025 Informatika a výpočetní technika
Téma:	On-line nástroj pro konverzi různých datových formátů On-line Tool for Converting Different Data Structures
Jazyk vypracování:	čeština

Zásady pro vypracování:

Cílem práce je implementovat on-line nástroj pro konverzi různých datových formátů jako jsou CSV, JSON, XML, XLS, SQL a další. Nástroj bude fungovat v on-line režimu nebo jako konverzní rozhraní (služba) mezi různými datovými zdroji.

1. Seznamte se s problematikou konverze různých datových formátů a SQL. Nastudujte rozdíly mezi různými verzemi jazyka SQL vzhledem k použité platformě (Oracle, MS SQL, MySQL, PostgreSQL a další).
2. Seznamte s existujícími nástroji pro konverzi datových formátů, datových pump a dalších souvisejících procesů.
3. Ve vhodně zvoleném prostředí proveďte analýzu návrh a implementaci vlastního on-line webového nástroje, který bude fungovat i jako služba pro konverzi datových formátů.
4. Funkčnost nástroje ověřte v praxi a zhodnoťte dosažené výsledky. Implementujte API rozhraní (Gloffer.com) pro jednu zvolenou platformu.
5. Porovnejte vaše řešení s existujícími systémy a navrhněte další možnosti rozšíření.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.
Web pro konverzi formátu CSV: http://www.convertcsv.com/

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Radoslav Fasuga, Ph.D.**

Datum zadání: 01.09.2015
Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka	prof. RNDr. Václav Snášel, CSc.
vedoucí katedry	*děkan fakulty*

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou/diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne: *26. dubna 2017*

David Kolb
.....
podpis studenta

Abstrakt

Tato práce řeší problematiku převodu různých datových struktur (souboru) mezi sebou. V práci jsou podrobně rozebrány jednotlivé datové formáty – CSV, XLS, XLSX, XML, JSON, YAML, NEON a SQL. U formátu SQL jsou stručně popsány nejrozšířenější databázové systémy – MySQL, Oracle, Microsoft SQL Server a PostgreSQL. V práci je dále uvedena problematika převodu a její řešení.

Pro implementaci převodníku jsem použil programovací jazyk Java, případně webový framework Spring MVC, webový server Apache Tomcat (verze 8) a několik externích knihoven pro převod jednotlivých formátů. Pro nasazení mnou naimplementované aplikace jsem zvolil webové služby Amazon AWS.

Vytvořené řešení umožňuje převod ze vstupního formátu CSV, XLS, XLSX, XML, JSON nebo SQL na výstupní formát CSV, XLS, XLSX, XML, JSON, SQL nebo YAML. Při převodu na SQL formát provede aplikace automatickou analýzu datových typů vstupu a tu potom zakomponuje do výstupním souboru.

Klíčová slova

Převod datových formátů, datová pumpa, převod datového formátu CSV, převod datového formátu XLS, převod datového formátu XLSX, převod datového formátu XML, převod datového formátu JSON, převod datového formátu YAML, převod datového formátu SQL, nasazení na Amazon AWS, využití webového serveru Apache Tomcat, implementace s frameworkem Spring MVC.

Abstract

This thesis deals with the issue of transformation variety data structures (files) among themselves. This work also describes datatype CSV, XLS, XLSX, XML, JSON, YAML, NEON and SQL in derail. Moreover, there is basic description for followed SQL database systems - MySQL, Oracle, Microsoft SQL Server a PostgreSQL. This thesis solves issue of data format transformation and its solution.

I have used programing language Java, it´s framework Spring MVC, Apache Tomcat (version 8) and few external JAR libraries, for implementation of this transformation application. The Amazon AWS web services were used to deploy the converter application implemented by myself.

Developed solution enables conversion of input data format CSV, XLS, XLSX, XML, JSON or SQL to output data format CSV, XLS, XLSX, XML, JSON, SQL or YAML. Within transformation to SQL output format is connected automatic analysis of data types in input file. This analysis is then incorporate to output file.

Key Words

Data formats transformation, data converter, datatype converter, data pump, CSV converter, XLS converter, XLSX converter, XML converter, JSON converter, YAML converter, SQL converter, Amazon AWS deployment, Apache Tomcat web server, implementation of Spring MVC framework.

Obsah

Seznam použitých symbolů a zkratek

CSV	- Comma Separated Values
XML	- Extensible Markup Language
XLS	- soubor vytvořen v programu Microsoft Excel do verze 2007
XLSX	- soubor vytvořen v programu Microsoft Excel od verze 2007
JSON	- JavaScript Object Notation
SQL	- Structured Query Language
YAML	- Yet Another Markup Language
TSL	- Tab-Separated Values
JAXP	- Java API for XML Processing
.exe	- Executable
BIFF	- Binary Interchange File Format
OLE 2.0	- Object Linking and Embedding 2.0
ZIP	- souborový formát pro kompresi a archivaci dat
OLE	- Object Linking and Embedding
SGML	- Standard Generalized Markup Language
HTML	- HyperText Markup Language
UTF-8	- 8-bit Unicode Transformation Format
UTF-16	- 16-bit Unicode Transformation Format
PHP	- Hypertext Preprocessor
DTD	- Document Type Declaration
SAX	- Simple API for XML
DOM	- Document Object Model
StAX	- Streaming API for XML
Enum	- Výčtový datový typ
T-SQL	- Transact-SQL (Structured Query Language)
PL/SQL	- Procedural Language/SQL (Structured Query Language)
PL/pgSQL	- Procedural Language/PostgreSQL
AWS	- Amazon Web Services
JSP	- JavaServer Pages
MVC	- Model-view-controller
URL	- Uniform Resource Locator
AWS	- Amazon Web Services
GB	- Gigabyte
MB	- Megabyte
RAM	- Random Access Memory
CPU	- Central Processing Unit
EC2	- Elastic Compute Cloud
SSH	- Secure Shell
DNS	- Domain Name System

Seznam ilustrací a tabulek

1 Úvod

Tato bakalářská práce se zabývá analýzou různých datových souborů a implementací nástroje pro jejich konverzi. Nástroj bude fungovat jako webová služba, na kterou bude uživatel moc přistupovat přes libovolný webový prohlížeč. Aplikace bude, mimo konverzi datových formátů mezi sebou, umožňovat exportování dat, z těchto datových formátů, do uživatelem vybraného formátu SQ. Výběr druhu SQL databáze je omezen na několik typů, které jsou na výběr v konečné aplikaci

Počáteční kapitola detailně popsuje nejběžnější a nejčastěji používané datové formáty. Je u nich popsána jejich vnitřní struktura, i aplikace, ve kterých se tyto formáty nejčastěji používají. Ve třetí kapitole jsou stručně popsány některé, z již existujících řešení problému. Jsou zde popsány online i desktopové aplikace, a jejich případné výhody i nevýhody. Čtvrtá kapitola se zabývá návrhem implementace mé online aplikace. V této kapitole je popsáno, jak by se měla aplikace chovat vzhledem k uživateli, a co vše je potřeba od získat uživatele, pro správný chod aplikace.

Pátá kapitola obsahuje podrobný popis implementace první části mého programu. Je zde popsáno, jak fungují jednotlivé metody aplikace převodníku, jak program postupuje od vstupu uživatele, až po výstupní, převedený formát. Poslední kapitola popisuje implementaci a nasazení parseru na server, tak aby mohl být přístupný online. Krom nasazení, je v kapitole popsáno uživatelské rozhraní a kroky, kterými musí uživatel projít, aby získal požadovaný výstupní datový formát. V závěru jsou shrnuty dosažené výsledky, omezení mé aplikace a problémy, kterým jsem musel čelit při implementaci a nasazení.

2 Formáty pro přenos dat

2.1 CSV

CSV neboli Coma-separated-values je základní souborový formát pro přenos 2D (plochých) dat. Díky své jednoduchosti se tento formát často používá pro přenos tabulkových dat mezi různými aplikacemi. Na jeho rozšíření má svůj podíl i aplikace Microsoft Excel, která umožňuje export dat právě do tohoto formátu. V dnešní době již není CSV formát průmyslovým standardem, právě kvůli nemožnosti vložení většího množství dat, případně informací o datech, do jednoho souboru. Ovšem i dnes se s tímto formátem můžeme setkat v jednoduchých aplikacích, které nevyžadují 3D reprezentaci dat.

V souborovém formátu CSV představuje jeden záznam jeden řádek. Pro ukončení řádků se používají různé symboly v závislosti na operačním systému. V operačních systémech Windows se používá znak pro nový řádek '/n' (ASCII/LF=0x0A), kdežto v unixových systémech se používají dva symboly, jeden pro návrat na začátek řádku '/r' a druhý nový řádek '/n' (ASCII/CRLF=0x0D 0x0A).

V ojedinělých případech se může stát, že je v poli zakomponován konec řádku (zalomení řádku), viz tabulka 1.

Tabulka 1 CSV pole se zalomením řádků

```
Pole 1:          Konferenční místnost 1
Pole 2:          Davide,
 zanes Petrovy dokumenty k revizi
 - M.T.
Pole 3:          10. 2. 2017
```

V těchto případech je potřeba takové pole vložit do dvojitých uvozovek, jako to je zobrazeno v tabulce 2. Nicméně mnoho aplikací pro zpracování CSV souborů nepodporuje zalomení řádku uvnitř pole, ani když je vloženo do uvozovek. Proto mohou být taková pole čtena nekorektně.

Tabulka 2 CSV pole se zalomením řádků ve správném formátu

```
Konferenční místnost 1, „Davide,
 zanes Petrovi dokumenty k revizi
- M.T.
“,10. 2. 2017
```

Jednotlivá pole se záznamy se oddělují pomocí takzvaného oddělovače, kde oddělovačem je většinou symbol čárky či středníku. V některých případech se jako oddělovač používá tabulátor, taková varianta se pak označuje zkratkou TSV, v češtině „tabulátorem oddělené hodnoty“. Veškeré záznamy mohou, ale nemusí být obaleny dvojitými uvozovkami. Ovšem v určitých případech jsou tyto uvozovky povinné.

Bílé znaky v podobě mezer a tabulátorů, které mohou být uvedeny před, nebo za záznamem, budou ignorovány. Například ze záznamu *David , Kolek* dostaneme výsledek *David, Kolek*. Pro zachování těchto mezer v textu lze použít dvojité uvozovky. Pokud chceme do určitého záznamu vložit text, který obsahuje čárku (oddělovač), musíme jej vložit do dvojitých uvozovek. Například text *David, Kolek* se vkládá jako „David, Kolek“. Pro použití uvozovek v textu je potřeba uvozovanou část textu doplnit o další pár dvojitých uvozovek. Příkladem může být tento text *"David Kolek ""Student"""*, kde je řetězec *Student* vložen do uvozovek. Ve výsledku text vypadá takto: *David Kolek „Student“*.

K jedné z nevýhod CSV formátu patří nemožnost automatického určení názvů jednotlivých sloupců. V obecném případě se považuje první řádek v souboru za hlavičku, která určí názvy sloupců, ale je prakticky nemožné automaticky zjistit, jestli první řádek opravdu obsahuje názvy sloupců, či nikoli. Tento problém se řeší vnějším zásahem, například dotazem na uživatele. V CSV formátu nechybí ani možnost vložení nulových hodnot. Ty se poznají podle toho, když za oddělovačem chybí hodnota a pokračuje další oddělovač, či končí řádek.

Příklad dat pro export do CSV formátu:

playerID	yearID	gameNum	gameID	teamID	IgID	GP	startingPos
aaronha01	1955	0	NLS195507120	ML1	NL	1.124	2
aaronha01	1956	0	ALS195607100	ML1	NL	1	
aaronha01	1957	0	NLS195707090	ML1	NL	1	9

Tabulka 3 data pro export do CSV formátu

Data převedená do CSV formátu:

```
playerID,yearID,gameNum,gameID,teamID,lgID,GP,startingPos
aaronha01,1955,0,NLS195507120,ML1,NL,1.124,2
aaronha01,1956,0,ALS195607100,ML1,NL,1,
aaronha01,1957,0,NLS195707090,ML1,NL,1,9
```

[1]

2.2 XLS a XLSX

Jak starší XLS, tak nové XLSX bylo vytvořeno za účelem použití v aplikaci Microsoft Excel. Soubor XLS se používal pro ukládání dat jednotlivých dokumentů (pracovních sešitů) programu Excel až do roku 2007, kdy vyšla nová verze a s ní byl představen nový XLSX formát. Největším rozdílem mezi těmito dvěma typy souborů je způsob ukládání dat. Pro uložení XLS se používá binární ukládací formát, kdežto novější XLSX bylo založen na formátu Office Open XML, který vychází z klasického XML. Informace v XLSX souboru tak nejsou ukládány binárně, ale jako text s použitím XML.

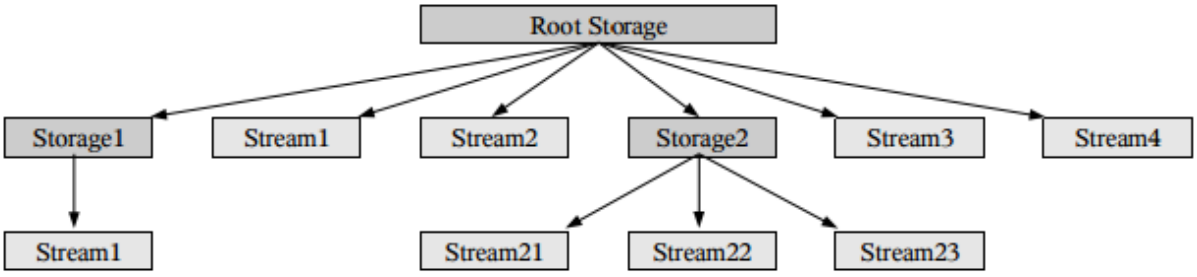
2.2.1 XLS

Jak již bylo zmíněno výše, XLS formát byl vytvořen společností Microsoft pro ukládání dat aplikace Excel. XLS byl používán až do roku 2007, kdy vyšla nová verze Microsoft Excel s novým formátem XLSX. Pro uložení dat ve formátu XLS se používá binární soubor BIFF. Během mnoha let vývoje aplikace Excel se vyvíjel i binární formát BIFF. Vyšlo hned několik verzí tohoto formátu a ty mezi sebou nejsou kompatibilní. Dodnes jsme se mohli setkat například s těmito verzemi souboru:

- Microsoft Excel 5.0 – BIFF 5 s příponou .xls
- Microsoft Excel 95 – BIFF 7 s příponou .xls
- Microsoft Excel 97, Microsoft Excel 2000, Microsoft Excel 2002 a Microsoft Office Excel 2003 – BIFF 8 s příponou .xls

2.2.1.1 Vnitřní struktura souboru

Tyto binární ukládací formáty jsou postaveny na formátu Compound Document File (také znám jako OLE 2.0 formát), což je v podstatě souborový systém v jednom souboru. Tato technologie se podobá souborovému systému FAT. Každý XLS soubor pak reprezentuje nový oddíl, jako tomu je například při rozdělení disku FAT systém.



Obrázek 1 Vnitřní uspořádání v OLE2

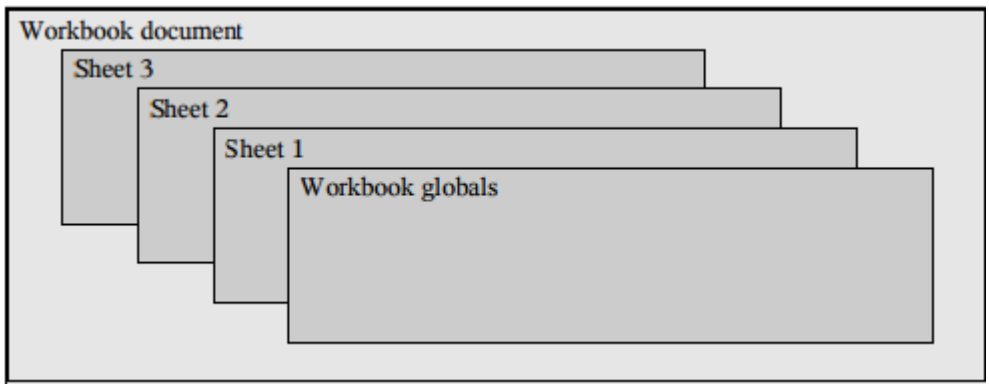
Uvnitř souboru nalezneme proudy (podobně jakou soubory v souborovém systému), ty jsou hierarchicky uspořádány do uložišť podobně jako složky a podsložky v souborovém systému. Všechna uložiště a proudy jsou pojmenovány. Přitom platí, že každé uložiště nebo proud musí mít jiný název než jejich předek. Každý Compound Document File v sobě obsahuje tzv. kořenové uložiště, které je buď přímým, nebo nepřímým rodičem všech dalších uložišť a proudů.

Stejně jako souborový systém, i OLE 2.0 využívá bloky (sektory) do kterých se ukládají proudy. Ty jsou alokovány a uvolňovány podle potřeby. Sektory mohou obsahovat jak uživatelská data, tak data potřebná pro organizaci souboru. Soubor se tak skládá z několika za sebou jdoucích sektorů, a také hlavičky (Compound Document Header), která je vždy na začátku souboru. Hlavička obsahuje všechna data, potřebná k práci se souborem – informace o umístění dalších klíčových bloků v souboru. Hlavička má vždy velikost 512 bytů a lze v ní určit několik údajů, jako je i velikost sektorů, která je defaultně 512 bytů.

2.2.1.2 Uspořádání dat v souboru

Všechny sektory spolu s hlavičkou pak tvoří takzvaný Workbook nebo Workspace (ve starších verzích Microsoft Excel to byl Worksheet).

Na níže uvedeném obrázku (Obrázek 2) lze vidět, že Workbook (pracovní sešit) se skládá z jednoho, až několika listů (Sheet). Jeden pracovní sešit lze poskládat z různých typů listů, od klasických textových, přes tabulkové, až po moduly Visual Basicu. Každý takový dokument však obsahuje tzv. Workbook Globals, které obsahuje globální nastavení pro daný pracovní sešit. Ve starších verzích Excelu nebylo možné mít v jednom dokumentu více listů než jeden. Proto se tehdy jako název používal tzv Worksheet nebo pracovní list.



Obrázek 2 Grafické znázornění Workbooku

Dokument typu Workspace (pracovní prostor) v sobě obsahuje jeden, až několik odkazů na pracovní sešity (Workbook), případně pracovní listy (Worksheet). Dá se tedy říct, že Workspace obsahuje hned několik různých souborů, s různým obsahem a typem. Součástí každého Workspace je i několik základních údajů o souborech v něm obsažených. Výchozí přípona dokumentu typu Workspace je „.xlw“.

Jak již bylo zmíněno dříve, v aplikaci Excel lze vytvářet různé typy listů, nejčastěji se používá klasický textový sešit. Ten se skládá podobně jako tabulka ze sloupců, řádků a buněk. Sešit může obsahovat maximálně 65536 řádků a až 256 sloupců. Do každé buňky pak lze vložit text, vzorec, datum nebo jiná data.

[2] [3]

2.2.2 XLSX

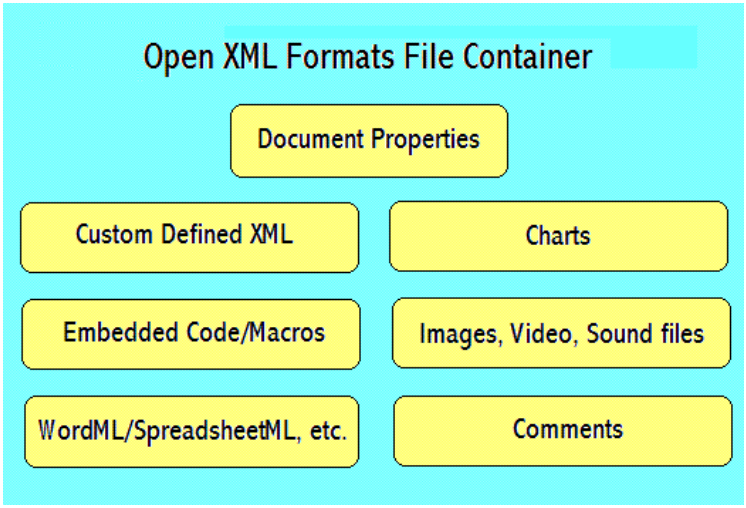
V návaznosti na příchod XML formátu v 90. letech minulého století začal i Microsoft pracovat na podpoře tohoto formátu pro své Office aplikace. Tuto podporu společnost nasadila ve verzi Microsoft Office 2000 a tehdy si uvědomila potřebu přechodu z binárních formátů na XML formát. Binární formáty sloužily dobře spoustu let, ale nebyly schopny zajistit některé z požadavků moderní doby, například snadný přenos dat mezi různými aplikacemi.

Právě proto byl v balíku aplikací Microsoft Office 2007 zaveden nový souborový formát Office Open XML Formats vycházející z klasického XML formátu. Vyřešily se tak problémy s přenosem dat mezi různým pracovním prostředím. Nový formát přináší vylepšenou zprávu, obnovení dat a práci s různými řádkovými firemními systémy. Formát umožňuje to, co binární formáty neumožňovaly, jakákoli aplikace s podporou XML formátu může přistupovat, případně pracovat s daty uvnitř souboru. Taková aplikace ani nemusí být z balíku Office nebo dokonce ani od Microsoftu. Snížila se i náročnost spojená s bezpečností. Souborový formát založen na XML, má svá data uložena jako text. Díky tomu mohou tato data bez problému projít firewallem a být analyzována.

2.2.2.1 Vnitřní struktura souboru

To, co vidíme v počítači jako jeden XLSX soubor, je ve skutečnosti hned několik souborů uložených v jednom kontejneru. Tento kontejner je založen na souborovém formátu pro kompresi ZIP. Uvnitř tohoto kontejneru nalezneme hned několik souborů s popisem dat, metadata a také data uložena uživatelem.

Většina těchto dat je uložena v XML. Data, která nejsou uložena v XML, mohou reprezentovat binární soubory – obrázky nebo OLE objekty použity v XLSX dokumentu. V kontejneru jsou obsaženy i části specifikující vztahy mezi jednotlivými soubory (částmi) XLSX souboru. Tyto vztahy popisují, jak se mají kombinovat a spolupracovat všechny části uvnitř kontejneru. Díky rozdělení dat a nastavení je možný snazší přenos dat mezi různými aplikacemi. Různé soubory formátu Office Open XML Formats v sobě mají spoustu různých částí, přitom některé z těchto částí zůstávají pro všechny, například metadata, média či vztahy. Jiné časti jsou pro každý typ aplikace unikátní, například pracovní sešit je unikátní pro Excel, kdežto slide je unikátní pro PowerPoint.



Obrázek 3 Příklad obsahu Open XML Formats kontejneru

Uspořádání dat uvnitř XML souboru je popsáno v následující kapitole 3.3, sekce vnitřní struktura souboru.

2.2.2.2 Uspořádání dat v souboru

Oproti staršímu XLS je v nové verzi pouze Workbook (pracovní sešit), do kterého můžete vložit jeden až několik listů (Sheet). Každý list v XLSX souboru představuje jeden soubor typu XML. Stejně jako ve starší verzi, i v této může Workbook obsahovat různé typy listů. Listy pak mohou obsahovat různé typy dat – čísla, data, text, obrázky a další. Oproti starší verzi se zde mění především rozsah listů, které mohou nyní obsahovat více než milion řádků a až 16,384 sloupců.

[4]

2.3 XML

XML neboli „Extensible Markup Language“ je textově zaměřený jazyk pro reprezentaci strukturovaných informací (dat, konfigurací, transakcí atd.). Byl odvozen ze staršího jazyku SGML (ISO 8879). Cílem tedy bylo vytvořit jazyk vycházející z SGML, tak aby mohl být snadno přenosný, přijatelný a zpracovatelný, podobně jako je tomu u HTML. XML byl navržen pro snadnou implementaci a vzájemnou spolupráci mezi SGML a HTML jazyky. [7] [8]

2.3.1 Vnitřní struktura a deklarace XML

Každý XML dokument musí dodržovat striktní syntaxi. Jednotlivé části dokumentu se rozdělují pomocí značek, které se nazývají elementy. Každá značka musí mít svůj začátek i konec, obdobně jako tomu je u HTML. Jazyk umožňuje zanoření jednotlivých elementů do sebe, čímž dokáže reprezentovat i 3D data. Kupříkladu uložení určité databáze do XML souboru – ten by se skládal z elementu databáze, ve kterém by bylo vnořeno několik elementů tabulka. V elementu tabulka by pak bylo vloženo několik elementů reprezentující jednotlivé záznamy tabulky (řádky). Každý element řádku by obsahoval další elementy pro jednotlivé položky (buňky) řádku. XML dokument se tak kaskádově dělí od největší části, v našem případě databáze, až po tu nejmenší část, buňku tabulky.

Elementy se v textu vyznačují pomocí tzv. tagů. Většině elementů odpovídají dva tagy – počáteční a ukončovací. Jednotlivé tagy se do dokumentu zapisují ve špičatých závorkách ‚<‘ a ‘>‘ příkladem takových tagů může být ***<database>*** (počáteční tag) a ***</database>*** (ukončovací tag). Ukončovací tag obsahuje před svým názvem lomítko ‚/‘.

Každý XML dokument musí mít tzv. kořenový element. Čili takový element, který obaluje celý dokument – všechny ostatní elementy jsou v něm vnořeny. Nemůže se stát, že by XML dokument obsahoval více kořenových elementů, v takovém případě by se jednalo o nekorektní zápis. Příkladem kořenového elementu může být element „report“ v níže zobrazeném obrázku 4.

```
<?xml version="1.0" encoding="UTF-8"?>
<report>
    <toc>
        <tocitem chapter="c1"/>
        <tocitem chapter="c2"/>
    </toc>
    <chapter id="c1" title="Summary">
        <sentence>This report describes the core concepts of XML, how to deploy it and
          its capabilities</sentence>
        <sentence>... </sentence>
    </chapter>
    <chapter id="c2" title="Description">
        <sentence> eXtensible Markup Language (XML) is a language subset of the Standard
       Generalized  Markup Language (SGML). SGML is a meta language, therefore, a </sentence>
        <sentence ref="something">... </sentence>
    </chapter>
</report>
```

Obrázek 4 Příklad XML syntaxe

Ne všechny elementy musí obsahovat nějakou hodnotu, mohou být nulové. Taková skutečnost se dá reprezentovat buď počátečním tagem, který je bezprostředně následován ukončovacím - ***<foto></foto>***, nebo lze použít jiný způsob značení prázdného elementu a to tím, že za název počátečního tagu uvedeme

lomítko - ***<foto/>***. Každý XML dokument musí pro všechny počáteční tagy obsahovat i jejich ukončovací tagy, případně zápis pro prázdný element s lomítkem. Pokud ukončovací tagy chybí, dokument je nekorektní.

Krom hodnot uvedených mezi jednotlivými tagy, mohou tagy obsahovat i tzv. atributy. Ty se používají k vložení upřesnění jednotlivých elementů nebo k přidání dalších informací. Každý atribut se skládá z názvu a hodnoty, jako tomu je u tagu *„tocitem“* na obrázku 4. Tento tag obsahuje atribut s názvem *„chapter“* a hodnotou *„c1“*. Za názvem atributu musí být uveden znak pro rovnítko a hodnota musí být uvedena ve dvojitých uvozovkách. Každý tag může mít několik atributů, které stačí oddělit mezerou. [11] [12]

Určité znaky (znaky syntaxe XML – markup) nelze zapsat jako hodnotu tagu, případně atributu. Pro takové případy zde existují tzv. znakové entity, které umožňují zapisovat tyto speciální znaky. V XML je přímo zabudováno jen pět takových znakových entit viz níže. Pokud by bylo potřeba definovat další znakové entity, lze definovat vlastní v definici typů dokumentu.

- **<** pro otevření špičaté závorky **<**
- **>** pro uzavření špičaté závorky **>**
- **&** pro ampersand **&**
- **"** pro znak dvojitých uvozovek **“**
- **'** pro znak apostrofu **'**

Pro definování speciálních znaků lze použít i hexadecimální hodnoty, například **&** pro zápis ampersandu.

2.3.1.1 Deklarace XML

XML dokument může, ale nemusí obsahovat deklaraci. V případě, že jí chceme v dokumentu uvést, musíme tak udělat na prvním řádku souboru. Nesmí jí předcházet žádný obsah, ani bílé znaky. V deklaraci se uvádí verze XML a použité kódování v dokumentu, viz ukázka deklarace výše. Zde je uvedena verze dokumentu 1.0 a kódování UTF-8

```
<?xml version="1.0" encoding="UTF-8"?>
```

Kódování dokumentu nemusí být v deklaraci obsaženo, ale pokud jej chceme uvést, musí být bezprostředně za verzí XML a musí obsahovat hodnotu existující znakové sady. V deklaraci může být uveden i atribut ***standalone***, který určuje, zda dokument existuje samostatně, nebo jestli dokument čerpá z externích zdrojů (souborů) a tím mění svůj obsah. Tento atribut může nabývat pouze hodnoty „yes“, což reprezentuje samostatný dokument a hodnoty „no“, která reprezentuje čerpání z externích zdrojů.

Pokud XML dokument neobsahuje žádné externí značkování, pak atribut *standalone* nemá žádný význam. Hodnota „yes“ se používá v případě, kdy externí značkování nijak neovlivní obsah dokumentu nebo způsob čtení XML souboru. V některých případech tak hodnota „yes“ může urychlit zpracování souboru, zamezí se totiž zbytečnému zpracování dalších souborů.

Pokud dokument obsahuje vnější značkování a atribut *standalone* není specifikován v deklaraci dokumentu, pak se defaultně nastavuje na hodnotu „no“. V takovém případě využívá externích dokumentů, ať už mění obsah XML nebo ne.

2.3.1.2 Kódování souboru

Jak už bylo uvedeno dříve, XML umožňuje deklarovat atribut pro kódování souboru, tento atribut není povinný, ale v některých případech je nezbytný. Pokud tento atribut v dokumentu neurčíme, použije jedno z defaultních kódováni, a to buď UTF-8, nebo UTF-16. Tato dvě kódování se odliší pomocí tzv. *Byte Order Mark*, které musí být uvedeno v každém XML dokumentu kódovaném v UTF-16. Změna kódování v atributu je potřeba především v těch případech, kdy dokument obsahuje „exotické znaky" a přitom není uložen v kódování UTF-8 (respektive UTF-16).

Takové případy nastávaly především v minulosti, kdy se například pro zobrazení českých znaků používalo kódování *windows-1250* nebo *ISO-8859-2*. V nynější době se již pro kódování českých znaků běžně používá kódování UTF-8. Může se však stát, že se setkáme se starším dokumentem s kódováním *windows-1250* v tom případě je do atributu *encoding* nutné uvést toto kódování. Případně celý soubor převést na kódování UTF-8 (UTF-16), pomocí nějaké speciální utility. Pokud toto neuděláme, speciální znaky se nám zobrazí nekorektně.

I když XML dovoluje použít jiné kódování než UTF-8 (respektive UTF-16), je vždy lepší použít právě jedno z těchto defaultních kódování. Hlavním důvodem, proč použít právě tato kódování je kompatibilita s dalšími aplikacemi. XML totiž nařizuje programům, které s ním pracují, aby byly schopné přečíst jak kódování UTF-8, tak UTF-16. Ovšem již nijak nenařizuje kompatibilitu s ostatními druhy kódování. Proto by se mohlo stát, že tyto aplikace přečtou soubor nekorektně.

2.3.1.3 Komentáře, procesní instrukce a sekce CDATA

Tak jako tomu je ve většině programovacích jazyků, tak i v XML nechybí možnost vytváření poznámek přímo v kódu. Komentáře začínají znaky „< ! −−" a jsou ukončeny znaky „−−>" (Obrázek 5), komentáře mohou být uvedeny na jakémkoli místě v XML soboru, nemohou však být uvedeny uvnitř tagu (například za názvem tagu).

```
<?xml version=`1.0`?>
<článek>
        <!-- toto je komentář, použití je stejné, jako v HTML...-->
        <název>XML pro web aneb od teorie k praxi, 2.díl</název>
        <autor>
        <?php
        if(!$autor && $input>0) {
                echo `...(autor nebyl zadán)...`;
        } else {
                echo $autor;
        }
        ?>
        </autor>
        <info typ="seriál" díl="2." periodicita="jednou týdně" />
        <domek název="Programování">
                <kategorie název="WAP / XML" />
        </domek>
        <příklad>
                <![CDATA[Příklad prázdného elementu v XML: <kategorie
název="WAP / XML" />...]]>
        </příklad>
</článek>
```

Obrázek 5 komentáře, procesní instrukce a sekce CDATA

XML umožňuje začlenění tzv. procesních instrukcí, většinou se jedná o segment kódu s příkazy libovolného programovacího jazyku. Takové příkazy mohou být zpracovány aplikací. Příklad takových procesních instrukcí lze vidět na obrázku 5, kde se jedná o instrukce jazyka PHP. Rozpoznání těchto instrukcí se provádí zavedením sekvence znaků „<?“ za kterou bezprostředně následuje identifikátor aplikace, která tyto instrukce zpracuje. Ukončení procesních instrukcí se značí sadou znaků „?>“. Aby nedocházelo ke konfliktům mezi procesními instrukcemi a deklarací XML, za počáteční sekvenci nelze uvést řetězec „xml“ v jakékoli kombinaci malých a velkých písmen. Tento řetězec je rezervován právě pro deklaraci dokumentu.

V případě potřeby začlenění velkého množství markup znaků do textu elementů, je možné použít tzv. sekci CDATA. Uzavřením textu do této sekce docílíme toho, při zpracování souboru nejsou markup znaky uvnitř brány v potaz. Sekce musí začínat řetězcem "<![CDATA[" a končit "]]>". [9]

2.3.1.4 Validita a DTD

Každý XML dokument splňující výše uvedené zásady syntaxe se nazývá *well-formed* neboli dobře utvořený. To ovšem ještě neznamená, že je takový dokument validní. XML obsahuje i tzv. DTD mechanismus (deklarace typů dokumentu) pro možnost automatické kontroly dokumentu. V DTD se deklarují jednotlivé elementy, které musí být obsaženy v XML dokumentu. Při deklaraci vždy určujeme jméno elementu a tzv. model obsahu. Ten určí, jaké další elementy může daný element obsahovat. [5]

```
<!ELEMENT faktura (odberatel, dodavatel, polozka+)>
```

Ve výše uvedeném příkladu můžeme vidět deklaraci elementu *faktura*, který v sobě musí obsahovat další elementy *odberatel*, *dodavatel* a několik elementů *polozka*. Správné určení modelu obsahu je důležité pro správnost celého XML dokumentu (jeho struktury). V určitých případech lze jako model obsahu použít klíčová slova **EMPTY** nebo **ANY**. Klíčové slovo **EMPTY** určuje, že element neobsahuje žádný obsah (ani další elementy), jako je tomu například u HTML tagu pro nový řádek <br>. Ten se dá deklarovat tímto způsobem:

```
<!ELEMENT br EMPTY>
```

V případě, že nechceme obsah elementu nijak omezit, lze použít klíčové slovo **ANY**. Toto klíčové slovo by se však nemělo používat příliš často, jelikož bychom se jeho použitím mohli zbavit kontroly syntaxe v XML dokumentu. Například kdybychom použili klíčové slovo ANY v každém elementu, neplatila by žádná pravidla pro strukturu a DPT by bylo zbytečné.

Model obsahu může obsahovat i složitější případy, například můžeme určit i přesné pořadí, v jakém mají být elementy uvnitř seřazeny. Celá modelová skupina se v tomto případě uzavře do závorek a jednotlivé elementy se oddělí čárkou v pořadí, jakém chceme.

```
<!ELEMENT clanek (nazev, (autor|editor), odstavec+)>
```

Znakem | (or, nebo) určíme, aby následoval pouze jeden z uvedených elementů. V příkladu výše bude následovat buď element *autor,* nebo *editor*. Při použití znaku | musíme všechny vybírané elementy uzavřít do závorek. Za každým elementem v modelové skupině lze uvést znak pro určení počtu možných výskytů tohoto elementu. Pokud žádný znak neuvedeme, pak zde element může být uveden jenom jednou. Znakem otazníku **?** řekneme, že je určený element nepovinný. Pro žádný až libovolný výskyt elementu použijeme znak hvězdičky *****. Znak + se využívá v situacích, kdy chceme alespoň jeden výskyt elementu.

Pokud element neobsahuje další elementy, ale jenom text, pak použijeme speciální řetězec **#PCDATA**. Může nastat i případ, kdy chceme v jednom elementu mít text i další elementy. Takovému případu říkáme *smíšený obsah* a značí se speciálním tvarem modelu obsahu, uvedený níže. V elementu *odstavec* tak můžeme psát jak text, tak používat další elementy *tučně* a *kurziva*.

```
<!ELEMENT odstavec (#PCDATA|tučně|kurzíva)*>
```

Krom deklarace elementů se do DTD vkládají i deklarace atributů obsažených v XML souboru. Oproti deklaraci elementu se zde nedá deklarovat každý atribut samostatně, ale deklaruje se hned celá skupina atributů pro určitý element. V podstatě se jedná o deklaraci atributů v určitém elementu. Syntaxe je podobná deklaraci elementů (viz obrázek 6).

```
<!ATTLIST info typ (seriál|článek|zpráva|PR) "článek"
                díl CDATA #IMPLIED
                periodicita CDATA #IMPLIED>
<!ATTLIST domek název CDATA #REQUIRED>
<!ATTLIST kategorie název CDATA #REQUIRED>
```

Obrázek 6 Deklarace atributů v DTD

Pro deklaraci se však používá jiné klíčové slovo *ATTLIST*, které je bezprostředně následováno názvem elementu, ke kterému patří daný seznam atributů. Dále následuje jméno atributu, jeho typ a defaultní hodnota. Jedna deklarace může obsahovat více atributů, tak jak tomu je například u deklarace elementu *info* v obrázku 6. Zde je deklarována hned trojice atributů – *typ, díl a periodicita*.

Atribut může nabývat hned několika typů:

- **CDATA** – pro libovolný text
- **ID** – atribut s tímto typem musí mít unikátní hodnotu (v celém dokumentu se smí vyskytnout 1)
- **IDREF** – atribut tohoto typu může obsahovat pouze hodnotu ID jiného atributu, jedná se tedy o cizí klíč k jinému atributu. Umožňuje se tak propojení v dokumentu.
- **IDREFS** – obdobný jako IDREF, s tím rozdílem, že zde může být uvedeno více hodnot jiných atributů, které jsou odděleny mezerou.
- **NMTOKEN** – omezuje hodnotu atributu na jedno slovo, skládající se z písmen a číslic.
- **NMTOKENS** – seznam jmen typu **NMTOKEN** oddělených mezerou
- **ENTITY** – musí obsahovat název externí neparsované entity
- **ENTITIES** – seznam jmen externích neparsovaných entit
- **Výčtový typ** – seznam konkrétních možných hodnot, které může daný atribut nabývat. Hodnoty výčtového typu musí splňovat podmínky NMTOKEN. Příklad výčtového typu naleznete na obrázku 6, atribut *typ*.

Výchozí hodnoty atributů lze deklarovat různými způsoby. Klíčové slovo **#REQUIRED** označuje atribut, který musí být v dokumentu vždy uveden, ale nemá žádnou výchozí hodnotu. Klíčový výraz **#IMPLIED** označuje takový atribut, který nemá žádnou výchozí hodnotu, a přitom nemusí být u určitého elementu uveden. Třetí možností je uvedení konkrétní defaultní hodnoty. V případě, kdy atribut nabývá vždy stejné hodnoty, se používá klíčový výraz **#FIXED**. Takový atribut se v XML dokumentu nemusí vůbec specifikovat.

Deklarace DTD lze vložit přímo do XML dokumentu, ale preferuje se ukládání do externího souboru s příslušnou příponou „.dtd“. Ten se poté připojuje k XML dokumentu pomocí deklarace typu dokumentu (DOCTYPE) níže uvedeným způsobem. Deklarace musí vždy obsahovat jméno kořenového elementu (*faktura*) a URL adresu k souboru s DTD (*faktura.dtd*).

```
<!DOCTYPE faktura SYSTEM "faktura.dtd">
<faktura>
  ...
</faktura>
```

Deklaraci DTD v externím souboru lze kombinovat spolu s deklarací v XML souboru, což umožní případnou změnu externího nastavení DTD. V případě konfliktu externího a interního DTD nastavení se totiž bere v potaz interní DTD.

O validaci XML dokumentu se pak stará parser, který zkontroluje celý XML soubor oproti DTD deklaraci, na kterou je odkazováno uvnitř souboru. Pokud XML soubor souhlasí s DTD deklarací, pak je takový dokument nejen dobře utvořený, ale i validní. [10] [13]

2.3.2 Práce s XML dokumentem

Jakožto moderní formát pro přenos dat jsou XML soubory každodenně využívány pro ukládání, přenos a čtení dat. Proto vznikla spousta nástrojů v různých programovacích jazycích, které umí tento formát zpracovat přečíst i zapisovat. Každý, kdo pracuje s těmito dokument,y by měl takové funkce (knihovny) preferovat před implementací vlastního parseru. Jednak je to rychlejší a také jsou tyto knihovny optimalizovány a laděny již několik let, takže budou v mnoha případech efektivnější.

Pro práci s XML dokumentem existují různé parsery, vždy se však dělí na dvě hlavní skupiny – SAX parsery a DOM parsery. Hlavním rozdílem mezi těmito dvěma přístupy k XML dokumentu je, že SAX čte soubor postupně, kdežto DOM si celý XML načte do paměti.

2.3.2.1 SAX (Push) parsery

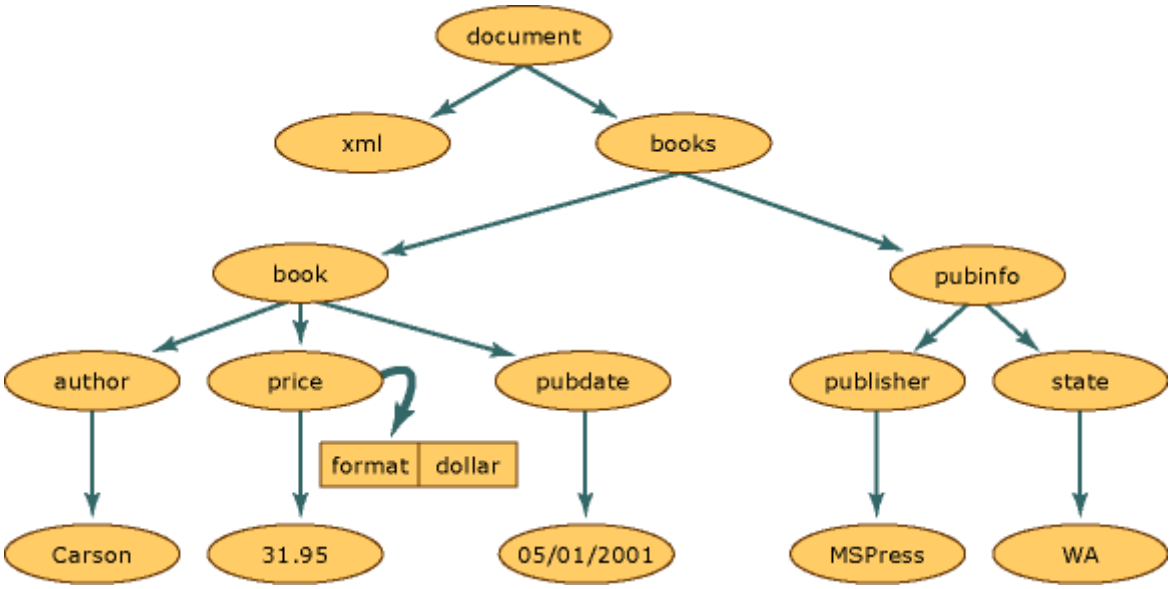
V přístupu SAX postupuje parser v XML dokumentu postupně, řádek po řádku. V případě, kdy narazí na nějakou důležitou událost, například narazí na entitu nebo atribut, zavolá obslužnou funkci, která nám umožní pracovat s těmito údaji. V paměti si uchovává minimální množství dat, takže paměť moc nezatěžuje. Díky tomu můžeme efektivně číst i velké XML soubory, aniž bychom potřebovali více paměti, než máme k dispozici.

Nevýhodou tohoto přístupu je zpracování XML souboru v jednom průchodu. To má za následek potřebu vytváření proměnných pro uložení sekvenčních dat. Takový přístup tedy vyžaduje vysokou logickou kapacitu programátora a není vhodný pro začátečníky, kteří by se mohli v takovém přístupu snadno ztratit. Další z nevýhod SAX je nemožnost upravovat XML dokument za chodu. V paměti totiž není celý dokument, takže lze pracovat pouze s jeho konkrétní částí. V tomto přístupu taky nevíme, jakou má dokument strukturu předtím, než jej celý přečteme.

Právě z důvodu nemožnosti okamžitého přecházení z jednoho prvku na jiný a nemožnosti úpravy dokumentu, je SAX přístup nevhodný pro takové aplikace, které tyto funkce vyžadují. Například webové prohlížeče nebo editory XML souborů.

2.3.2.2 DOM parsery

Na rozdíl od SAX parseru si DOM parsery načtou celý XML soubor do paměti v podobě stromové struktury (Obrázek 7). Načtení dokumentu do paměti sice přináší nároky na procesorový čas, a ještě větší nároky na paměť, ale na druhou stranu umožňuje lepší a pohodlnější práci s XML souborem. Uložení souboru v paměti nám umožňuje použít funkce, které v SAX přístupu nejsou možné. XML můžeme editovat, případně vytvořit nový XML dokument.



Obrázek 7 Stromová struktura XML v paměti

Díky tomu, že známe celou stromovou strukturu XML dokumentu, můžeme přistupovat přímo k těm prvkům, které zrovna potřebume. Právě díky možnosti využití těchto funkcí se DOM parsery používají v prohlížečích, kde je potřeba okamžitý přístup k určitému prvku soboru a jeho případná modifikace.

2.3.2.3 StAX (Pull) parsery

StAX neboli Streaming API for XML je interface pro čtení a zápis XML souborů, vytvořen komunitou programovacího jazyka Java. K jeho vytvoření vedla omezení obou klasických přístupů k XML souborům, a to již zmíněný DOM a SAX přístup. StAX se snaží vzít něco od obojího a může se tak řadit mezi tato dvě řešení. Stejně jako SAX si neuchovává celý strom XML souboru v paměti, ale čte jej postupně. V souboru postupuje pomocí kurzoru, kterým určuje, kde přesně se v souboru zrovna nacházíme.

Aplikace (programátor co jí píše) prochází pomocí tohoto kurzoru souborem, přesně podle toho, jak to to jeho řešení vyžaduje. Kurzor se v souboru pohybuje vždy od začátku do konce souboru, a to v jakýkoli čas, kdy si to aplikace vyžádá. Kurzor ukazuje vždy na konkrétní věc v XML dokumentu, na atribut, začátek elementu, komentář, začátek souboru atd. Kurzor se vždy pohybuje dopředu, takže se v souboru nelze vracet k předchozím prvkům. Obvykle se parser pohybuje dokumentem prvek po prvku.

Tady nastává rozdíl mezi SAX, který chrlí informace ze souboru a vyžaduje, aby se aplikace postarala o všechny události, které při průchodu nastanou. Takový přístup k datům, kde si program říká, co potřebuje, vychází právě z DOM parseru, kde procházíme strom v paměti a ptáme se na určité informace. StAX navíc umožňuje čtení i zápis nových XML souboru najednou.

V některých případech však StAX lze využít jen velmi obtížně a neúčinně. Stejně jako SAX prochází tento parser souborem postupně. Takže se nehodí všude tam, kde se nehodí SAX, při průchodu souboru si totiž musíte sami vytvořit strukturu souboru, abyste věděli, jak vlastně XML ve skutečnosti vypadá. V nejhorších případech se může stát, že je tato struktura velmi složitá a komplexní stejně jako originální dokument. V takových případech se vyplatí použít právě DOM parsery, jelikož získáte náhodný přístup k prvkům, který je přehlednější a mnohdy jednodušší.

StAX se hodí především tehdy, když potřebuje zpracovat velké XML soubory, a přitom potřebujete postupně projet kousek po kousku celým dokumentem.

[6] [14]

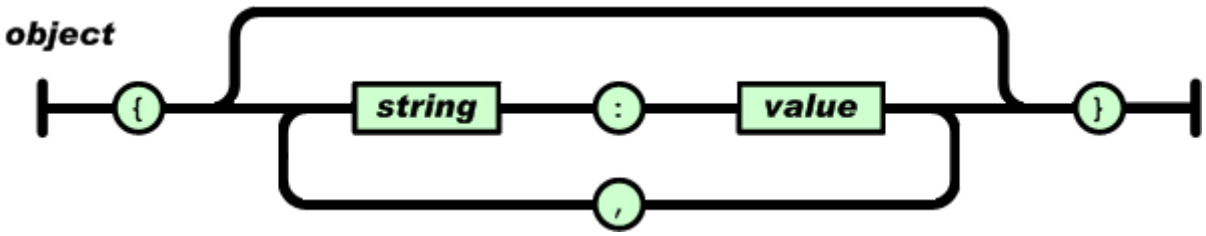
2.4 JSON

JSON je textově založený formát pro výměnu dat mezi různými aplikacemi. Pro svůj zápis používá jednoduchou syntaxi vycházející z programovacího jazyka JavaScript. Díky tomu je snadno čitelný jak programem, tak člověkem. Díky své jednoduchosti si vysloužil velikou oblíbenost v použití webových aplikací. Oproti XML formátu je tedy jednodušší, ale není tak výkonný při zápisu rozsáhlých dokumentů.

Text JSON dokumentu se skládá z různých posloupností Unicode znaků, které splňují danou syntaxi. Syntaxe JSON obsahuje šest řídících (strukturálních) znaků – hranaté závorky [] pro pole objektů, složené závorky {} pro objekt, dvojtečku a čárku. Dále je zde možné uvést jeden z trojice konstantních znaků – *true*, *false* a *null*. Před každým strukturálním znakem je možno uvést jeden z podporovaných bílých znaků: mezera, tabulátor, konec řádku '/n' a návrat kurzoru na začátek '/r'.

2.4.1 JSON objekt a jeho hodnoty

Základním stavebním kamenem JSON souboru je JSON objekt, který je nutnou součástí každého správně formátovaného dokumentu. JSON objekt je neuspořádaná množina páru klíč a hodnota. Takovýto pár můžeme znát z různých programovacích jazyků jako strukturu, hashovací tabulku, slovník atd. Objekt vždy začíná levou složenou závorkou {, ta je následována názvem objektu, dvojtečkou a hodnotou objektu, celý objekt je pak ukončen pravou složenou závorkou }.



Obrázek 8 Grafické znázornění JSON objektu

Hodnotou JSON objektu mohou být tyto datové typy

- **String** – sekvence Unicode znaků (0 až několik), která musí být umístěna do dvojitých uvozovek. Taková sekvence může obsahovat jakýkoliv Unicode znak, krom řídících a znaku, bílých znaků a znaků pro dvojité uvozovky, takové znaky se musí značit espace znakem – obraceným lomítkem \, například \" pro uvedení dvojitých uvozovek do textu.
- **Number** – čísla jsou v JSON reprezentována v klasické desítkové soustavě, můžeme vkládat jak záporná, desetinná (s tečkou), tak čísla s exponentem. Exponent musí být značen znakem malé *e* nebo velké *E*, ten může být následován symbolem pro plus/mínus a až deseti číslicemi. Znaky pro určení nekonečna nebo neurčené hodnoty čísla nejsou podporovány.
- **Boolean** – logická hodnota *true* nebo *false*
- **Null** – hodnota reprezentující prázdnou hodnotu *null*

Pokud potřebujeme vložit jiný datový typ, než jsou podporované hodnoty JSON objektu, pak je musíme převést a vložit do jedné podporované. Klasickým příkladem může být datum, které můžeme do JSON uložit jako text.

Krom těchto základních hodnot může JSON objektu nabývat i dalších dvou hodnot, a to je JSON objekt a JSON array. Pomocí těchto dvou hodnot je JSON schopen reprezentovat i 3D data, jelikož může obsahovat další vnořený objekt, nebo pole objektů. Každý vnořený JSON objekt může nabývat různých typů hodnot (String, Number, Boolean, Null, JSON objekt nebo JSON array).

JSON Array (pole)

Je zvláštní hodnotou JSON objektu, která obsahuje seřazenou kolekci hodnot daného datového typu (jeden z datových typů uvedených dříve). Pole se značí hranatými závorkami, kde levá závorka značí začátek [a pravá konec pole]. Jednotlivé hodnoty v poli musí splňovat pravidla pro svůj datový typ (viz datové typy). JSON Array může samozřejmě obsahovat i kolekci JSON Objektů. Jednotlivé hodnoty pole se pak stejně jako objekty oddělují čárkou, která může být následována bílým znakem.

Soubor typu JSON se klasicky označuje příponou *.json*. Korektnost souboru je zajištěna tehdy, když soubor začíná znakem levé složené závorky a končí znakem pravé složené závorky (soubor začíná JSON objektem), nebo pokud soubor začíná znakem levé hranaté závorky a končí znakem pravé hranaté závorky (soubor začíná JSON Array) a zároveň splňuje všechny dříve zmíněná pravidla pro zápis JSON objektu.

Příklad validního JSON kódu:

```
{
	"name": "John",
	"age": 30,
	"cars": [{
		"name": "Ford",
		"models": ["Fiesta", "Focus", "Mustang"]
	}, {
		"name": "BMW",
		"models": ["320", "X3", "X5"]
	}, {
		"name": "Fiat",
		"models": ["500", "Panda"]
	}]
}
```

Ve výše uvedeném kódu můžeme vidět příklad jednoho JSON objektu, který může reprezentovat záznam zákazníka půjčovny vozidel. V tomto příkladu nalezneme sérii párů klíč a hodnota, kde klíč „cars“ obsahuje pole JSON objektů reprezentující název výrobce automobilu a model auta. Klíč „models“ je zvláště zajímavý tím, že obsahuje další pole hodnot typu String.

[15] [16]

2.5 YAML

YAML neboli YAML Ain't Markup Language je textový formát pro snadnou serializaci strukturovaných dat. Tento formát nabízí jednoduchost, která schází v některých případech složitému XML. Neobsahuje složité konstrukce, jako jsou v XML začátky / konce tagů, escape znaky a podobně. Přitom zachovává možnost ukládání 3D (strukturovaných) dat. Dokáže tedy vyjádřit i složitější konstrukce, jako jsou struktury a pole. Ačkoli není tento formát velmi známý, práci s ním zvládneme skoro v každém programovacím jazyku. Pro práci s YAML soubory bylo implementováno spousta knihoven v různých programovacích jazycích například pro Javu, C#, C/C++, PHP a spousta dalších.

Tvůrci YAML si dali za úkol vytvořit formát, který by byl snadno pochopitelný a čitelný pro lidi, přitom byl přenosný mezi různými programovacími jazyky a podporoval ukládání jejich datových struktur. Dále bylo potřeba zajistit konzistentní model, tak aby formát podporoval generické nástroje. Díky podpoře zpracování při jednom průchodu, má YAML možnost zpracovat i rozsáhle soubory.

2.5.1 YAML a JSON

Oba dva formáty mají za cíl být srozumitelné a čitelné pro lidi. Ovšem každý z těchto formátů má jiné priority. JSON se snaží být co nejvíce jednoduchý a univerzální, tak aby byl co nejsnáze generovatelný a zpracovatelný, díky tomu může být zpracován každým moderním programovacím prostředím. YAML má jako nejpřednější cíl uživatelskou přívětivost a podporu pro serializaci libovolných nativních datových struktur. Díky tomu je YAML sice velmi snadno uživatelsky čitelný, ale o něco složitější při vytváření a parsování.

Společné vlastnosti těchto dvou formátů dělají z YAMLu podmnožinu formátu JSON, která nabízí ještě lepší čitelnost a dokonalejší informační model. Tento vztah mezi formáty JSON a YAML platí samozřejmě i v praxi, každý validní soubor ve formátu JSON je totiž zároveň validní YAML dokument. Právě toto zajišťuje snadnou možnost migrace mezi těmito dvěma formáty, v případě že vám nestačí použití JSON, ale potřebujete komplexnější YAML.

Vzhledem k odlišným specifikacím mezi těmito formáty však není možnost migrace stoprocentně zaručena. Formát JSON totiž ve své specifikaci jen doporučuje unikátnost klíčů při mapování. Kdežto YAML tuto unikátnost vyžaduje, právě proto je migrace mezi těmito dvěma formáty možná jen v tom případě, kdy jsou klíče v JSON souboru unikátní.

2.5.2 Vnitřní struktura YAML souboru

Jak již bylo zmíněno v předchozích kapitolách, YAML se snaží pojmout co nejvíce datových struktur z různých programovacích prostředí. Právě proto obsahuje několik základních stavebních kamenů (základních prvků), které lze do souboru zapsat.

2.5.2.1 Sekvence

Nejjednodušším prvkem v YAML souboru je Sekvence, ta reprezentuje prostý seznam hodnot, jaký můžeme znát z reálného světa, například seznam jmen, knih, filmů atd. Pro každou položku ze seznamu se používá nový řádek. Položka se značí uvedením symbolu pomlčky před požadovanou hodnotu, příklad takto zapsané sekvence je uveden níže.

```
- Mark McGwire #CEO
- Sammy Sosa
- Ken Griffey
```

Data v sekvenci mohou být v libovolném formátu, ať už se jedná o desetinné číslo, datum nebo text. Výše uvedený příklad se skládá ze tří textových prvků. Pokud bychom takovýto soubor načetli do svého programu, mohli bychom k němu přistupovat stejně jako k prvkům pole. Důležitá vlastnost sekvence je ta, že zachovává pořadí záznamů. Mřížka předcházena mezerou v příkladu ‘ #’ reprezentuje začátek poznámky (komentáře) v YAML, ta končí koncem řádku.

2.5.2.2 Mapa

Dalším prvkem v podstatě stejně složitým jako Sekvence je Mapa. Ta reprezentuje dvojici provázaných hodnot – klíč a hodnota, jinak známo jako asociativní pole či slovník v programovacích jazycích. Klíč musí být unikátní a identifikuje konkrétní záznam v Mapě. Příklad jednoduché YAML mapy je uveden níže.

```
hr:  65    # Home runs
avg: 0.278 # Batting average
rbi: 147   # Runs Batted In
```

Mapa se vyznačuje jednoduchou syntaxí, kde název klíče je následován dvojtečkou, ta musí být bezprostředně následována mezerou. Poté už může následovat hodnota v libovolném formátu. Stejně jako v Sekvenci, tak i Mapě představuje jeden řádek jeden záznam.

Složený klíč

Krom jednoduchého klíče uvedeného v předchozím příkladu můžeme použít i klíč složený, a to konkrétně ze Sekvence. Takový klíč se značí symbolem otazníku na začátku řádku, poté následuje mezera a samotný klíč, v tomto případě Sekvence. Tu značíme buď klasicky s pomlčkami, nebo pomocí závorek (zápis uveden v odstavci Sekvence Sekvencí). Příklad níže ukazuje, jak takový složený klíč vypadá (obsahuje oba způsoby zápisu).

```
? - Detroit Tigers
  - Chicago Cubs
:
  - 2001-07-23

? [ New York Yankees,
    Atlanta Braves ]
: [ 2001-07-02, 2001-08-12,
    2001-08-14 ]
```

2.5.2.3 Kombinace Map a Sekvencí

Krom základních formátů mohou Mapy a Sekvence obsahovat i složitější datové struktury, a to buď Mapu nebo Sekvenci. Přitom se musí splnit výše uvedená pravidla pro zápis Sekvencí a Map.

Příklad Sekvencí v Mapě:

```
american:
  - Boston Red Sox
  - Detroit Tigers
  - New York Yankees
national:
  - New York Mets
  - Chicago Cubs
  - Atlanta Braves
```

Příklad Map v Sekvenci:

```
-
  name: Mark McGwire
  hr:   65
  avg:  0.278
-
  name: Sammy Sosa
  hr:   63
  avg:  0.288
```

Sekvence Sekvencí se mohou značit dvojím způsobem, buď klasickým způsobem s pomlčkami, kde první pomlčka bez hodnoty následována bílým znakem značí začátek další vnořené Sekvence. Položky této vnořené sekvence musí být vždy na samostatném řádku, odsazeny o dvě mezery více než předchozí úroveň Sekvence, viz příklad níže.

```
-
  - full name
  -
    - name
    - surname
  - hr
  - avg
```

Druhým, přehlednějším způsobem je zápis pomocí hranatých závorek, kde jednotlivé položky oddělíme čárkami. Stále platí pravidla pro zápis Sekvencí, řádek začíná pomlčkou, ta je následována mezerou a začátkem hranaté závorky. Uvnitř závorky pak následují hodnoty libovolného typu, které jsou vzájemně odděleny čárkou bezprostředně následovanou mezerou. Konec záznamu se značí pravou hranatou závorkou. Příklad takového zápisu je uveden níže.

```
- [full name,    [name, surname], hr, avg  ]
- [Mark McGwire, [Mark, McGwire], 65, 0.278]
- [Sammy Sosa,   [Sammy, Sosa],   63, 0.288]
```

Pro **zápis Mapy Map** pak můžeme využít syntaxi se složenými závorkami či klasickou syntaxi s dvojtečkou. Zde dodržíme zásady pro utváření Mapy tak, že zadáme klíč, dvojtečku a poté hodnotu. Hodnota je zde další mapa, tím pádem ji obklopíme složenými závorkami a jednotlivé Mapy uvnitř závorek pak oddělíme čárkou, bezprostředně následovanou mezerou. V příkladu níže jsou uvedeny dva způsoby zápisu Mapy Map, a to buď zápis na jeden řádek, nebo zápis na více řádků.

```
Mark McGwire: {hr: 65, avg: 0.278}
Sammy Sosa: {
    hr: 63,
    avg: 0.288
  }
```

V případě klasického zápisu s dvojtečkou si povedeme stejně. Na začátek řádku vložíme klíč, dvojtečku a na další řádek vložíme vnořenou mapu odsazenou o dvě mezery od předchozí úrovně.

2.5.2.4 Dokumenty

Jeden YAML soubor může obsahovat několik částí, dalo by se říct záznamů. Jednotlivé části se oddělují třemi po sobě jdoucími pomlčkami „---“ a nazývají se Dokumenty (viz níže). Začátek každé takové části se značí právě těmito pomlčkami umístěnými na začátek řádku. Na tomto řádku není vložena žádná hodnota. Většina souborů ve formátu YAML začíná takto, případně je zde ještě uvedena poznámka za symbolem mřížky. Při posílání dat komunikačním kanálem lze využít sekvenci tří teček „…“ na konci Dokumentu, ta značí konec záznamu, aniž by začal nový záznam, nebo byl uzavřen komunikační kanál.

```
--- # Ranking of 1998 home runs
- Mark McGwire
- Sammy Sosa
- Ken Griffey
…


# Team ranking
---
- Chicago Cubs
- St Louis Cardinals
```

Opakované záznamy

Pro zaznamenání opakovaných záznamů v YAML formátu můžeme použít tzv. kotvu, která je značena symbolem ampersand ’&’, za ampersand poté vložíme název kotvy, na kterou budeme poté odkazovat. V příkladu níže je kotva značena *“&SS*”, za kotvou se potom vkládá hodnota (záznam), kterou chceme někde opakovat “*Sammy Sosa*”. Poté jíž můžeme na kotvu odkazovat symbolem hvězdičky ‘*’ za ten se napíše název odkazované kotvy.

```
---
hr:
  - Mark McGwire
  # Following node labeled SS
  - &SS Sammy Sosa
rbi:
  - *SS # Subsequent occurrence
  - Ken Griffey
```

Toto je spíš ale takový výstřelek, lze totiž použít klasický zápis, kde se nám budou záznamy (řádky) opakovat.

2.5.2.5 Odstavce textu

Formát YAML umožňuje zapisovat delší odstavce textu, a to se zachováním formátování či bez něj. Jednotlivé odstavce textu můžeme ukládat například do Mapy, takže použijeme klíč, dvojtečku, mezeru a jednu ze čtyř možností uchování textu.

```
accomplishment: >
  Mark set a major league
  home run record in 1998.
```

Symbol pravé špičaté závorky '>' určuje, že za ním uvedený odsazený text tvoří odstavec. YAML parser spojí všechny řádky, přebytečné mezery a text se uloží do souvislého textu. Prázdné řádky se vyhodnocují jako konec odstavce a začátek nového.

Při použití symbolu '|' bude každý řádek brán jako odstavec, takže výše zmíněný kód bude tvořit dva odstavce. Samotný symbol rovné čáry vynechává prázdné řádky, ty jako odstavce nebere. Proto je zde možnost kombinace toho symbolu se symbolem pro plus '+' (**Keep**), nebo minus '-' (**Strip**). Kombinace s plusem „|+" zachovává všechny řádky v záznamu (odstavci) tak jak jsou, to znamená, že v posledním řádku odstavce ponechává ukončení řádku a bere v potaz i prázdné řádky. Naopak kombinace s minusem „|-" u posledního řádku odstavce odstraní ukončení řádku a prázdné řádky nebere v potaz.

```
Accomplishment1: |
  Mark set a major league


Accomplishment2: |+
  Mark set a major league


Accomplishment3: |-
  Mark set a major league
```

První text tedy bude obsahovat jeden symbol pro ukončení řádku, druhý příklad bude tyto symboly obsahovat dva a třetí příklad nebude obsahovat žádný.

[17] [18]

2.6 Neon

Datový formát Neon je téměř totožný s výše popsaným formátem YAML. Největší rozdíl mezi těmito dvěma formáty je v tom, že Neon podporuje takzvané „entity“, takže může být snadno použit pro parsování anotací phpDocku (standart dokumentace programovacího jazyka PHP). Dalším rozdílem je použití bílého znaku „tabulátor“ pro odsazení jednotlivých příkazů/dat. Neon je oproti YAMLu o něco jednodušší a jeho zpracování probíhá rychleji. Tento datový formát se v aktuální době využívá výhradně ve spojení s programovacím jazykem PHP a frameworkem Nette vytvořených společností Nette Foundation, kterou založil český vývojář David Grudl. Mezi jeho nevýhodu však patří to, že tento formát není standardizován. Formát tak nemá platnou specifikaci ani dokumentaci, ze které by se dala čerpat přesná struktura a posloupnost příkazů pro zápis tohoto formátu.

Příklad dat ve formátu NEON:

```
# my web application config

php:
    date.timezone: Europe/Prague
    zlib.output_compression: yes  # use gzip

database:
    driver: mysql
    username: root
    password: beruska92

users:
    - Dave
    - Kryten
    - Rimmer
```

[32]

2.7 SQL

SQL neboli Structured Query Language, je základní jazyk určen pro práci nad relačními databázemi. Jazyk byl vytvořen v 70. letech minulého století a je četně používán databázovými administrátory i vývojáři aplikací pro nejrůznější platformy. Jazyk SQL může být použit k vytváření a jiným operacím s databázovou tabulkou, do které se vkládají data. Jazyk nám kromě vytváření tabulek a sloupců umožňuje různé operace na všemi daty nebo jejich částí, kterou můžeme specifikovat vybranými příkazy tohoto jazyka. O standardizaci jazyka SQL se zasloužily organizace ISO a ANSI. Od svého stvoření se tento standard vyvíjel a byla vydána hned řada revizí.

Vzhledem k tomu, že standard jazyka SQL neobsahuje některé potřebné konstrukce pro implementaci některých funkcí, tvůrci různých databázových systému si je museli vytvořit sami. Z tohoto důvodu je přenositelnost mezi jednotlivými databázovými systémy skoro nemožná. Rozdíly mezi jednotlivými systémy lze vidět především v procedurálním rozšíření, ale najdou se i v základní syntaxi. Četné rozdíly jsou i v definici, množství a nastavení jednotlivých datových typů

V této bakalářské práci nás bude zajímat především vytváření databáze, tabulek, sloupců a plnění jednotlivých dat do těchto sloupců. Proto se budu zabývat spíše popisem příkazu „CREATE“ a „INSERT“.

Příkaz CREATE:

Tento příkaz slouží v databázových systémech pro vytvoření databáze i tabulek. Posloupnost při vytváření databáze je velmi jednoduchá. Při vytváření uvedeme řetězec „CREATE DATABASE“ a za něj vložíme jméno vytvářené databáze. Vytváření jednotlivých tabulek je pak o něco složitější. Na začátku uvedeme řetězec „CREATE TABLE“, poté uvedeme jméno vytvářené tabulky a do závorek začneme postupně vkládat jednotlivé sloupce tabulky. Vždy se uvádí název sloupce, poté datový typ, a nakonec jeho integritní omezení. Datové typy a jejich integritní omezení je často mezi různými databázovými systémy rozdílné. Níže uvedený příklad reprezentuje posloupnost vytváření tabulky v databázovém systému MySQL.

```
CREATE TABLE Student (
    sID int PRIMARY KEY,
    jmeno VARCHAR(20) NOT NULL,
    rok_narozeni INT
);
```

Příkaz INSERT:

Tento příkaz slouží ke vkládání jednotlivých dat do databázových tabulek, jak můžeme vidět na níže uvedeném příkladu. Obecný SQL INSERT se skládá z několika nezbytných části. První z nich je uvedení řetězce „INSERT INTO“, poté následuje název tabulky, do které chceme vkládat. Do závorek se pak vkládají jednotlivé atributy, které reprezentují názvy sloupců. Dále musí následovat řetězec „VALUES“, za kterým se v závorkách uvedou vkládané hodnoty pro určené sloupce. Hodnoty se mohou vkládat buď s uvozovkami, nebo bez nich, formát vkládaní je vždy specificky určen konkrétním databázovým systémem. [19]

```
INSERT INTO Tabulka (Atribut1, ... AtributN)
VALUES (Hodnota1, ..., HodnotaN)
```

2.7.1 MySQL

Databázový systém MySQL je jeden z nejpopulárnějších open source databázových systému na světě. Prvenství si zasloužil svou výkonností, spolehlivostí a jednoduchou syntaxí. Těmito vlastnostmi se MySQL stal nejlepší volbou pro webově založené aplikace. Mezi příklady takových aplikací patří Facebook, Twitter, YouTube a mnoho dalších. [20]

CREATE TABLE

Pro vytvoření tabulky se v MySQL využívá již dříve uvedená definice. Za klíčovým slovem pro vytvoření tabulky lze využít řetězec „IF NOT EXISTS“, ten zajistí, že se tabulka nebude vytvářet, pokud již existuje (lépe řečeno, nevyhodí chybu o existenci tabulky). Definice může být doplněna i o různá integritní omezení. Mezi ty nejčastěji používané patří:

- NOT NULL – určuje nemožnost vložení nulové hodnoty do sloupce
- NULL – umožňuje vložit nulovou hodnotu
- DEFAULT výchozí_hodnota – umožňuje nastavení výchozí hodnoty sloupce
- PRIMARY KEY – určuje, že daný sloupec (sloupce) tvoří primární klíč tabulky
- UNIQUE – hodnoty v daném sloupci se nemohou opakovat
- AUTO_INCREMENT – daná hodnota sloupce se vytváří automaticky
- CONSTRAINT – nastavení omezení pro daný sloupec
- **FOREIGN KEY** (název_klíče) **REFERENCES** odkazovaná_tabulka(odkazovaný_sloupec) – vytváří cizí klíč s odkazem na klíč(klíče) jiné tabulky [21]

Specifikace jednotlivých datových typů pro databázový systém MySQL naleznete v příloze „Specifikace datových typů“, která je součástí této bakalářské práce.

INSERT INTO

Pro vkládání jednotlivých dat se využívá již zmíněný řetězec „INSERT INTO“. V MySQL lze vkládat hned několika způsoby, mezi ty nejčastější patří výše zmíněný příklad obecného vložení dat do tabulky. Dále se často využívá vložení bez opakujících se klíčových slov „INSERT INTO“ a „VALUES“, jak je tomu na níže uvedeném příkladu.

```
INSERT INTO `data` (`totalItems`, `startIndex`, `itemsPerPage`, `updated`)
VALUES
( "800", "1", "1", "2010-01-07T19:58:42.949Z"),
( "124", "1", "10", "2012-09-13T06:54:12.874Z"),
( "245", "1", "10", "2015-15-21T14:25:36.124Z");
```

Tímto způsobem se při vytváření vstupního (inicializačního) souboru do databáze ušetří spousta řádku. Pro každý vkládaný záznam minimálně 1 řádek. Dalším způsobem je vložení do tabulky, bez vyjmenování sloupců, do kterých se mají data vkládat. Zde však musíme uvést vždy data pro všechny sloupce dané tabulky (krom automaticky generovaných), jinak se vložení nezdaří. [22]

2.7.2 Oracle

Databázový systém Oracle byl navržen společností Oracle Corporation, jakožto vysoce výkonná databáze s širokou podporou. Pro zefektivnění běhu a vývoje některých aplikací bylo do tohoto databázového systému přidáno procedurální rozšíření s názvem PL/SQL. Toto rozšíření umožňuje efektivnější práci s jednotlivými záznamy, odchyt výjimek databáze, vytváření kurzorů a další. Hlavní výhodou je umístění rozšíření přímo v databázovém systému. Aktuální verze systému je 12c, ten je nově od této verze umístěn v cloudu a funguje tedy jakou cloudová služba. Díky tomu se minimalizuje potřebný čas pro vytvoření vlastní databáze. [23]

CREATE TABLE

Posloupnost příkazů při vytváření databázové tabulky je naprosto totožná jako v předchozím případě MySQL, jsou zde však jiná integritní omezení i jiné podporované datové typy. Níže uvedený seznam reprezentuje nejpoužívanější integritní omezení:

- NOT NULL – určuje nemožnost vložení nulové hodnoty do sloupce
- UNIQUE – hodnoty v daném sloupci se nemohou opakovat
- CONSTRAINT – nastavení omezení pro daný sloupec
- CONSTRAINT název_omezení **PRIMARY KEY** (název_ sloupce/ů) – určuje, že daný sloupec (sloupce) tvoří primární klíč tabulky
- CONSTRAINT název_omezení **FOREIGN KEY** (název_klíče) **REFERENCES** odkazovaná_tabulka(odkazovaný_sloupec) – vytváří cizí klíč s odkazem na klíč(klíče) jiné tabulky
- CHECK – kontroluje vkládané prvky podle vloženého pravidla [24]

Specifikace jednotlivých datových typů pro databázový systém Oracle naleznete v příloze „Specifikace datových typů“, která je součástí této bakalářské práce.

INSERT INTO

Pro vložení dat do tabulek se využívá stejný řetězec „INSERT INTO“, jako v předchozích případech. V Oracle systému lze vkládat dvěma způsoby, první z nich je již zmíněný příklad obecného vložení dat do tabulky. Druhým je hromadné vložení dat do jedné nebo i více tabulek najednou. Výhodou tohoto vložení je, že se příkaz provede jako celek, a ne postupně jeden příkaz za druhým. Příklad hromadného vložení:

```
INSERT ALL
  INTO suppliers (supplier_id, supplier_name) VALUES (1000, 'IBM')
  INTO suppliers (supplier_id, supplier_name) VALUES (2000, 'Microsoft')
  INTO suppliers (supplier_id, supplier_name) VALUES (3000, 'Google')
SELECT * FROM dual;
```

[25]

2.7.3 Microsoft SQL Server (MS-SQL)

Ze stejných důvodů, jako vytvořila společnost Oracle, vytvořila svůj databázový systém i velmi dobře známá společnost Microsoft. Tento databázový systém se většině případů využívá s technologiemi svého výrobce. Ačkoli byl SQL Server vždy úzce spjat s operačním systémem Windows, a tím nemohl přímo konkurovat databázi Oracle, nyní se chystá verze i pro platformu Linux [26]. Stejně jako konkurence, tak i Microsoft do svého databázového systému přidal procedurální rozšíření s názvem T-SQL. Toto rozšíření umožňuje efektivnější práci s jednotlivými záznamy, odchyt výjimek databáze, vytváření kurzorů a dalších funkcí.

CREATE TABLE

Pro vytvoření databázové tabulky v Microsoft SQL Serveru se využívá téměř totožná posloupnost, jako v případě MySQL, integritní omezení a datové typy jsou však rozdílné [27]. Níže uvedený seznam reprezentuje nejpoužívanější integritní omezení:

- NOT NULL – určuje nemožnost vložení nulové hodnoty do sloupce
- NULL – určuje možnost vložení nulové hodnoty do sloupce
- UNIQUE – hodnoty v daném sloupci se nemohou opakovat
- CONSTRAINT – nastavení omezení pro daný sloupec
- CONSTRAINT název_omezení **PRIMARY KEY** (název_sloupce/ů) – určuje, že daný sloupec (sloupce) tvoří primární klíč tabulky
- CONSTRAINT název_omezení **FOREIGN KEY** (název_klíče) **REFERENCES** odkazovaná_tabulka(odkazovaný_sloupec) – vytváří cizí klíč s odkazem na klíč (klíče) jiné tabulky
- IDENTITY – používá se pro klíče, daná hodnota sloupce se vytváří automaticky
- CHECK – kontroluje vkládané prvky podle vloženého pravidla [28]

Specifikace jednotlivých datových typů pro databázový systém Microsoft SQL Server naleznete v příloze „Specifikace datových typů“, která je součástí této bakalářské práce.

INSERT INTO

Pro vložení dat do tabulek se využívá stejný řetězec „INSERT INTO“, jako v předchozích případech. Databázový systém SQL Server nabízí hned několik možností vložení záznamů. První z nich je již zmíněný příklad obecného vložení dat do tabulky. Druhým je hromadné vložení dat do konkrétní databázové tabulky. Toto vložení je téměř totožné jako tomu je u databázového serveru MySQL, jsou zde použity jiné uvozovky. Také přináší jisté zjednodušení a zkrácení kódu o přebytečné údaje [25]. Příklad hromadného vložení:

```
INSERT INTO status (id, reason, value)
VALUES
( 0, 'limitedSyndication', 'restricted'),
( 1, 'unlimitedSyndication', 'not restricted');
```

V SQL serveru lze vkládat i prázdné hodnoty, které budou nahrazeny defaultními hodnotami daných sloupců (ty musí být zadány), viz příklad níže:

```
INSERT INTO employees
(employee_id, last_name, first_name)
DEFAULT VALUES;
```

2.7.4 PostgreSQL

PostgreSQL, zkráceně Postgres je jedním z dalších bezplatných databázových systému relačního typu pro komerční i nekomerční využití. Tento systém lze provozovat na všech předních platformách – Windows, Linux, Unix. Přitom splňuje požadavky na moderní databázový systém, díky podpoře všech základních příkazů a integraci datových typů JSON a XLM. PostgreSQL splňuje základní certifikaci SQL jazyka (ANSI) a je k ní zdarma dostupná rozsáhlá dokumentace všech funkcí, která je přehledná a nachází se v ní příklady použití daných příkazů. Databázový systém umožňuje spuštění procedur, napsaných v různých programovacích jazycích (Perl, Python, C) nebo ve svém speciálně navrženém jazyku PL/pgSQL vycházejícím z PL/SQL společnosti Oracle [29].

CREATE TABLE

Pro vytvoření tabulky se v Postgresu využívá dříve uvedená definice. Za klíčovým slovem pro vytvoření tabulky lze využít řetězec „IF NOT EXISTS“, toto zajistí zamezení chybových hlášek při pokusu o opakované vytvoření již existující tabulky. Definice může být doplněna i o různá integritní omezení. Mezi ty nejčastěji používané patří:

- NOT NULL – určuje nemožnost vložení nulové hodnoty do sloupce
- NULL – umožňuje vložit nulovou hodnotu
- CHECK – kontroluje vkládané prvky podle vloženého pravidla
- DEFAULT výchozí_hodnota – umožňuje nastavení výchozí hodnoty sloupce
- PRIMARY KEY – určuje, že daný sloupec (sloupce) tvoří primární klíč tabulky
- UNIQUE – hodnoty v daném sloupci (hodnota spojených sloupců) musí být unikátní
- AUTO_INCREMENT – daná hodnota sloupce se vytváří automaticky
- CONSTRAINT – nastavení omezení pro daný sloupec
- **FOREIGN KEY** (název_sloupce) **REFERENCES** odkazovaná_tabulka (odkazovaný_sloupec) – vytváří cizí klíč s odkazem na klíč (klíče) jiné tabulky [30]

Specifikace jednotlivých datových typů pro databázový systém PostgreSQL naleznete v příloze „Specifikace datových typů“, která je součástí této bakalářské práce.

INSERT INTO

Pro vkládání jednotlivých dat se klasicky využívá řetězec „INSERT INTO“. Vkládat lze několika způsoby, jeden ze způsobů je již zmíněné obecné vložení dat do tabulky. Dalším způsobem je hromadné vložení dat, to je obdobné jako v případě MySQL databáze nebo Microsoft SQL Server viz příklad níže.

```
INSERT INTO status (id, reason, value)
VALUES
( 0, 'limitedSyndication', 'restricted'),
( 1, 'unlimitedSyndication', 'not restricted');
```

Stejně jako v předchozích systémech, tak i v tomto se tímto způsobem zkrátí a usnadní zápis při vkládání dat v inicializačním souboru databáze. Dalším způsobem je vložení do tabulky, bez vyjmenování sloupců, do kterých se mají data vkládat. Zde však musíme uvést vždy data pro všechny sloupce dané tabulky (krom automaticky generovaných), jinak se vložení nezdaří. [31]

3 Existující aplikace pro konverzi dat

3.1 Email2DB (ThinkAutomation)

Jedná se o nástroj pro převádění dat prostřednictvím mailových zpráv. Získaná data použije pro aktualizaci databáze. Tato data zpracovávající aplikace nabízí i jiné funkce, například nastavení triggerů a jiných akcí nad databází. Tato funkčnost umožňuje uživateli krom získání dat ze zprávy i reagovat nad obsahem emailu a okamžitě provádět akce s daty. Tyto možnosti aplikace mohou být užitečné například při přidávání uživatelů do seznamu zákazníků, nebo dokonce i k přijímání objednávek od zákazníků prostřednictvím mailových zpráv. Tento systém ovšem není volně přístupný, pokud jej chce někdo využívat, musí zaplatit nemalou částku za licenci. Společnost vydávající tento software ovšem nabízí zkušební verzi programu na 30 dní zdarma. [39]

3.2 Log Parser

Log Parser je opensource utilita ovládaná pomocí konzole. Pomocí příkazů dokáže zpracovávat různé datové soubory například CSV, XML, logovací soubory, Active Directory a další. Log Parseru pak sdělíte, co a jak chcete zpracovávat a jaký chcete výstup. Výstupem může být různě formátovaný textový soubor, nebo specifický datový formát, například SQL, SYSLOG nebo tabulka. Ovládání nástroje však vyžaduje znalost příkazů tohoto programu a schopnost ovládání příkazové konzole v operačních systémech Microsoft Windows. Jedná se tedy spíše o nástroj pro pokročilejší uživatele. Výhodou tohoto řešení je však jeho otevřená licence, tudíž za něj uživatel nemusí platit. [40]

3.3 Log Parser QL

Log Parser QL je jednou z dalších opensource řešení pro zpracování datových souborů. Dokáže zpracovávat formát CSV a jiné datové formáty s oddělovači. Po otevření souboru s daty Vám Log Parser QL zobrazí popis jednotlivých ve zpracovaném souboru. Po načtení polí z datového souboru, lze nad nimi provádět SELECT operace a vybrat si tak potřebná data přímo ze souboru. Takto vybraná data pak mohou být zobrazena na obrazovku počítače, nebo uložena do jiného datového souboru, a to buď do CSV, nebo HTML souboru. Toto řešení je implementováno jako desktopová aplikace, která je uživatelsky přívětivější než konzolová. Uživatel jí může ovládat prostřednictvím různých výběrových menu. Aplikace je stejně jako Log Parser volně šiřitelná a uživatel za ní nemusí platit. [41]

3.4 Data Parse Free Edition

Data Parse Free Edition je volně šiřitelná JAVA aplikace pro analyzování dat. Aplikace je velice flexibilní, ale není určena pro běžné uživatele, je totiž ovládaná prostřednictvím skriptovacího jazyka. Výhodné je ovšem intuitivnost tohoto jazyka, který lze celkem pohotově ovládat. Aplikace je tedy určena spíše pro programátory, kteří chtějí provádět pokročilejší operace nad daty. Vstupem tohoto programu pak může být text nebo hex soubory. Výstupem tohoto softwarového řešení je pak soubory ve formátech CSV, XML, HTML a další. Grafické rozhraní aplikace je velice jednoduché a omezené,

dokáže pouze vybrat příslušný zdrojový soubor a určit kde se má uložit cílový soubor. Zbylá funkčnost je dosažena prostřednictvím již zmíněného skriptovacího jazyka. [42]

3.5 Convert CSV

Webová aplikace Convert CSV umožňuje svým návštěvníkům převod z datového formátu CSV (případně Excel) na jiný datový formát. Příkladem výstupního datového formátu je JSON, XML, YAML nebo SQL. Aplikace umožňuje nahrání vlastního souboru, nebo vložení textu pro převod do pole. Oproti ostatním zmíněným nástrojům má Convert CSV pokročilejší možnosti pro nastavení jak vstupního, tak výstupního datového souboru. Krom nástrojů pro převod z CSV datového souboru nalezneme na webu i odkaz na aplikaci umožňující převod různých datových formátů mezi sebou, například XML na JSON, YAML na JSON atd.[37]

3.6 Code Beautify

Code Beautify je online aplikace s širokou škálou nástrojů. Nabízí jak možnost převodu různých datových formátů mezi sebou, tak další funkce, ke kterým patří například formátovač kódu, validátor datových formátů nebo i editor. Výhodou této online aplikace je hlavně jeho bezplatnost, ale i snadné použití a široká základna funkcí. Má však i svá omezení, například převodník z jednoho datového formátu na druhý nepodporuje zpracování souborů větších než 2 MB. [36]

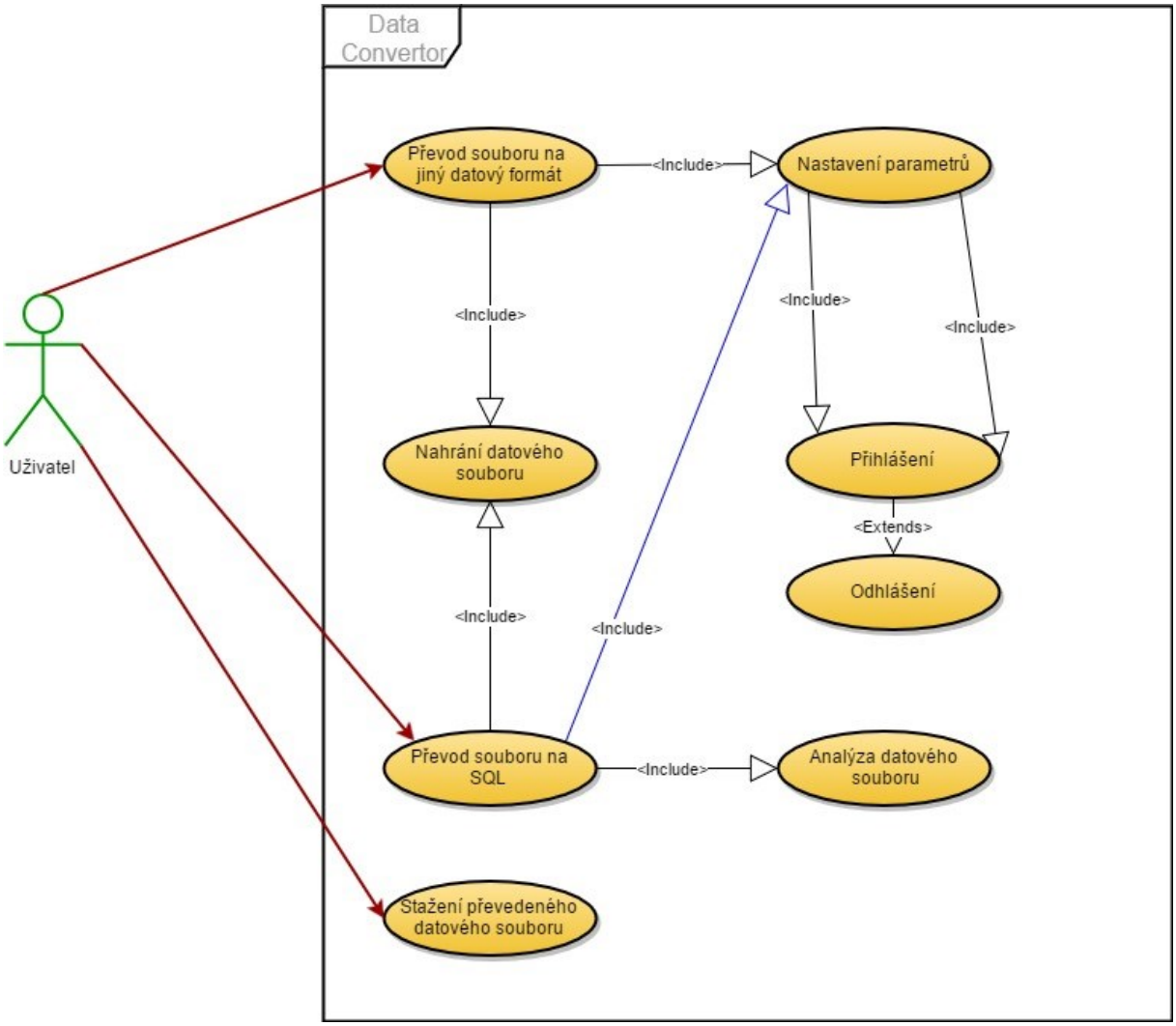
3.7 FREEFORMATTER

Toto online řešení nabízí konverzi mezi datovými formáty XML, JSON a CSV. Krom těchto převodníků nabízí i formátovač různých datových formátů a další užitečné nástroje. Nástroje pro převod z jednoho formátu na jiný fungují téměř stejně, jako tomu bylo u online řešení Code Beautify. Uživatel nahraje vstupní soubor, který je dále převeden na určitý výstupní soubor. Přitom je zde i pole, do kterého může místo vstupního souboru vložit vstupní text. Při převodu CSV na XML může uživatel vložit XML Template soubor. Všechny převodníky pak umožňují základní nastavení formátování výstupu. Omezení vstupních souboru je zde nastaveno taktéž na 2 MB. [38]

4 Návrh implementace

Jak již bylo zmíněno v úvodu bakalářské práce, aplikace bude fungovat jako online nástroj. Pro snadnější a univerzálnější použití aplikace jsem se rozhodl rozdělit implementaci do dvou částí. První část je implementace samotného parseru, který umožní konverzi vstupního datového souboru na jiný (popřípadě stejný) výstupní soubor. Druhá část (aplikace) se pak bude starat o práci s uživatelem, a to prostřednictvím online rozhraní.

V níže uvedeném Use Case diagramu můžete vidět, jak by měl fungovat výsledný program s webovým rozhraním, po spojení výše uvedených dvou částí.



Obrázek 9 Use Case diagram pro registrovaného uživatele

Obrázek 9 se zabývá situací uživatele při přístupu k online rozhraní aplikace. Uživatel může přistoupit k aplikaci a konvertovat svůj vstupní datový soubor, v jednom z podporovaných formátů, a převést jej na jiný (případně stejný) výstupní datový soubor, který tato aplikace podporuje. Pro převod vstupního souboru na jeden z výstupních si aplikace od uživatele vyžádá nastavení několika parametrů. Prvním parametrem aplikace je vstupní soubor, který je potřeba nahrát prostřednictvím online rozhraní na server, kde se aplikace nachází. Doba nahrávání vstupního souboru může trvat několik sekund až několik

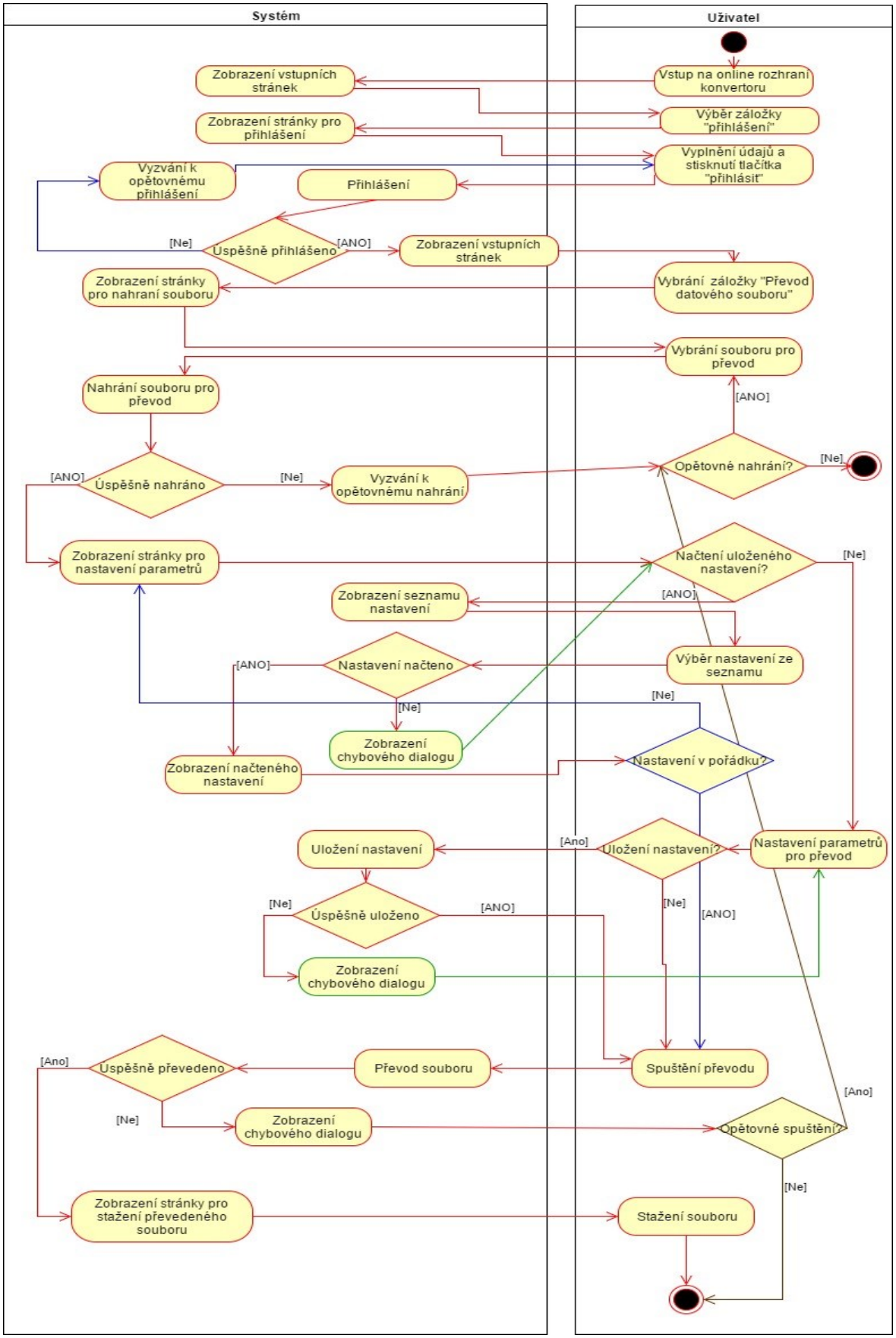
minut v závislosti na velikosti vstupního souboru a datového připojení uživatele. Po úspěšném nahrání vstupního souboru může uživatel pokračovat v dalším kroku a to je nastavení parametrů.

Mezi tyto parametry může patřit jméno výstupního souboru, typ vstupního souboru, typ výstupního souboru. V případě zvolení převodu na SQL datový soubor si aplikace vyžádá několik dalších parametrů, tak aby konverzi provedla správně.

Aplikace počítá s dvěma typy uživatelů – registrovaný a anonymní. Registrovaný uživatel bude mít možnost uložení, případně načtení, nastavení parametrů z minulého použití aplikace a spuštění parseru s tímto nastavením. Toto nastavení musí být samozřejmě korektní vůči vstupnímu datovému souboru. Anonymní uživatel nebude mít možnost uložení ani načtení nastavení.

Oba uživatelé budou moct převádět datové soubory ve stejném rozsahu. Po zdárném převodu ze vstupního souboru na výstupní, budou mít oba uživatelé možnost stáhnutí výstupního souboru do svého zařízení. Přístup k výstupnímu souboru bude pro oba uživatele limitován v rámci aplikace, po jejím zavření (ukončení) již nebude mít uživatel možnost tento soubor stáhnout. Vstupní i výstupní soubor však zůstane po dobu určenou administrátorem na serveru. Po specifikované době budou oba soubory smazány.

Detailní postup registrovaného uživatele, webovou aplikací můžete vidět na obrázku 10, kde je zobrazen diagram aktivit.



Obrázek 10 Diagram aktivit uživatele s přihlášením

5 Implementace aplikace parseru

Pro implementaci parseru jsem zvolil programovací jazyk Java. Tento jazyk jsem zvolil z důvodu zjednodušení problematiky. Oproti jazyku C++ lze v Javě využít více knihoven, ať už interních nebo externích, což podstatně zjednodušuje práci programátora. Tento jazyk jsem zvolil také pro jeho snazší nasazení jakožto webové aplikace. Pro implementaci programu pro převod datových souborů jsem použil hned několik externích knihoven, které budou uvedeny a popsány v dalších odstavcích.

Vzhledem k tomu, že vstupem programu je datový soubor typu XML, CSV, XLS, XLSX nebo JSON a jeho výstupem může být jakýkoli z těchto formátů, pro přímý převod mezi těmito formáty by bylo potřeba implementovat 20 různých převodníků. Proto jsem zvolil cestu mezi-formátu a tím jsem si výrazně snížil množství potřebných převodníků. Vstupní formáty se tedy převedou právě do tohoto formátu, poté se provede analýza prvků, a nakonec se tento mezi-formát převede na jeden z výstupních formátů. Díky tomu mi stačilo naimplementovat převodníky ze vstupních souborů na tento univerzální a další převodníky z tohoto univerzálního formátu na výstupní.

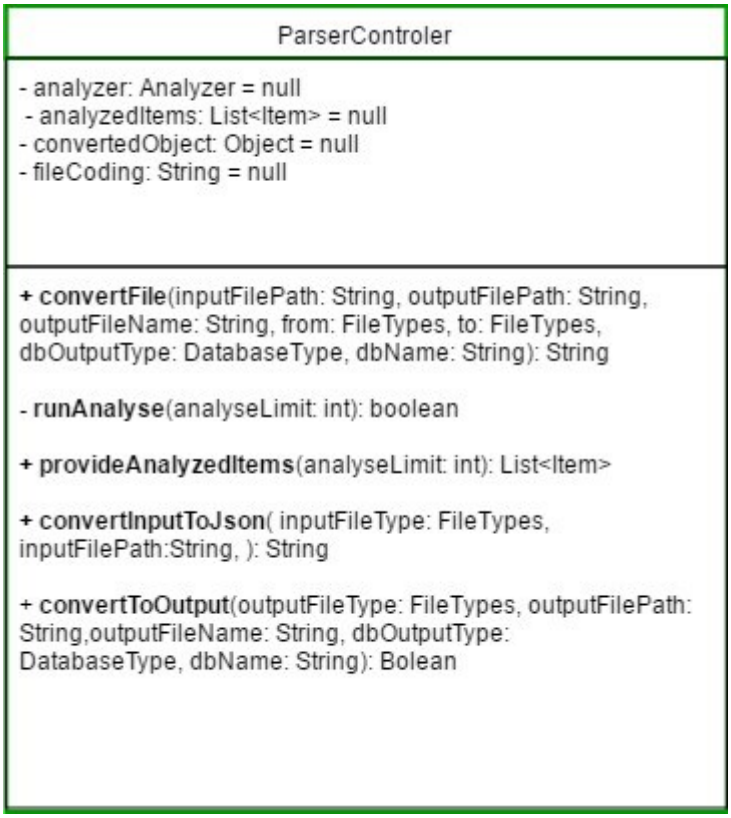
Nemusím tak vytvářet převodník pro převod z určitého vstupního na určitý výstupní soubor. Mezi-formát mi také umožňuje přidání nového převodníku pro další vstupní formát, aniž bych musel zasahovat do funkčnosti analýzy či převodníků pro výstup. Toto samozřejmě platí i při přidání nových převodníků pro další výstupní formát.

Pro tento mezi-typ jsem zvolil formát JSON a to z toho důvodu, že umí reprezentovat 3D data, je široce podporován v nejrůznějších knihovnách jazyka JAVA a oproti XML je podstatně jednodušší. Detailní popis formátu JSON, XML a dalších naleznete v předchozích kapitolách této práce.

5.1 Zpracování vstupu programu

Funkčnost celého programu parseru je řízena třídou s názvem „ParserControler“ (obrázek 11). Při vytvoření instance této třídy má programátor přístup k veřejným metodám „converFile“, „provideAnalyzedItems“, „convertInputToJson“ a „convertToOutput“, první z nich má za úkol převést vstupní soubor na výstupní, a to prostřednictvím zadaných parametrů. K tomu třída využívá ostatních metod veřejných i privátních metod. Funkce „convertInputToJson“ se stará o převod ze vstupního formátu na již zmíněný mezi-formát, „convertToOutput“ zajišťuje převod z mezi-formátu na výstupní formát.

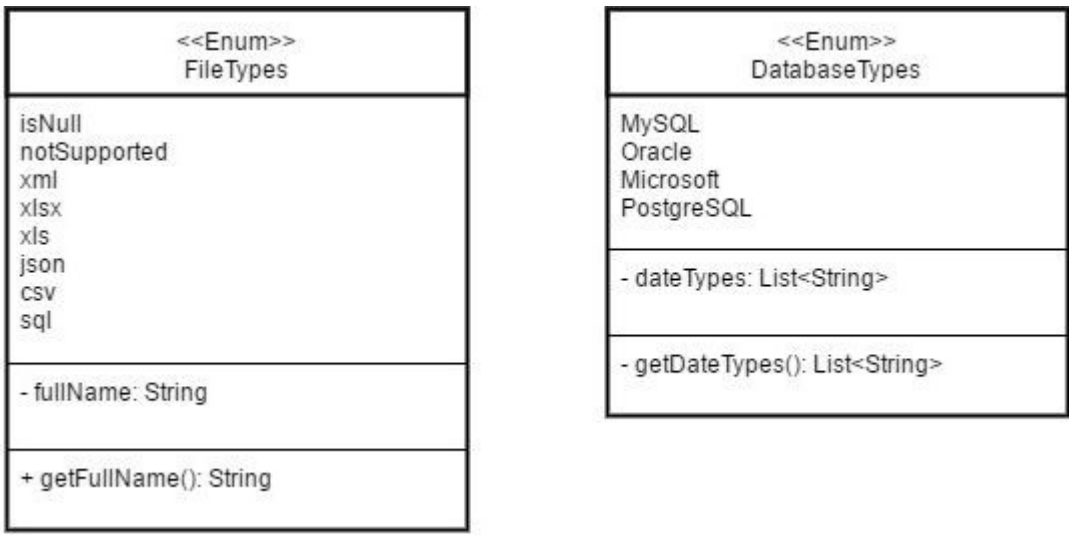
V této kapitole nás bude zajímat právě metoda pro zpracování vstupu na mezi-formát a veřejná metoda „convertFile“, která tuto metodu spouští. Veřejná metoda při svém volání vyžaduje hned několik parametrů. Prvním parametrem je plná cesta ke vstupnímu souboru, včetně jeho jména a příslušné koncovky.



Obrázek 11 Diagram třídy ParserControler

Druhý parametr je cesta, do které se má vytvořit výstupní soubor (vytváření cesty je popsáno v kapitole nasazení na web). Třetí parametr je název pro výstupní soubor, včetně příslušné koncovky pro daný výstupní formát, který je určen pátým parametrem. Čtvrtý parametr specifikuje datový typ vstupního souboru. Další parametr této metody je typ databáze, tento parametr se používá v případě, že uživatel vybere převod do výstupního formátu SQL. Poslední parametr je název pro vytvářenou databázi.

Datové typy uvedené na obrázku 12 jsou buď klasický řetězec (String), nebo mnou vytvořené datové typy. Mnou vytvořené typy jsou „FileTypes“ a „Database Type“, oba dva jsou typu Enum.



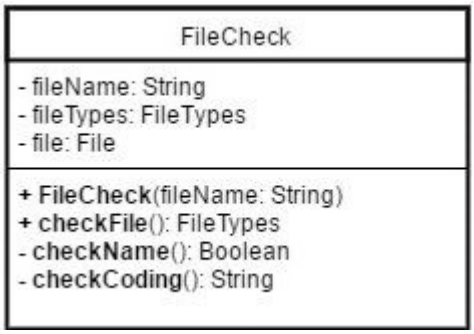
Obrázek 12 Třídní diagram pro databázové a typové Enum

Datový typ FileTypes obsahuje seznam všech podporovaných vstupních a výstupních formátů. Dále obsahuje i položku „isNull“, která se používá v případě, že je vstupní soubor prázdný. Položka „notSupported“ se používá v případě, že uživatel vloží jako vstup nepodporovaný vstupní formát. Obě položky jsou použity především pro interní použití programu. Toto Enum má dále i privátní konstantu „fullName“ a veřejnou metodu k získání hodnoty této konstanty. Ta reprezentuje plný (uživatelsky přívětivější) název hodnoty Enum. Například pro elegantnější zobrazení uživateli – XLS se zobrazí jako Old Excel – XLS.

Datový typ DatabaseTypes obsahuje seznam všech podporovaných typů databází, do kterých lze vstupní převést. Privátní konstanta „dateTypes“ a veřejná metoda k získání její hodnoty reprezentuje pole řetězců specifikující podporované datumové typy jednotlivých databází. Tento seznam datumových typů se používá při vytváření SQL insertu (SQL výstupního formátu).

Metoda „converFile“ v sobě dále používá instanci třídy „FileCheck“ a její veřejnou metodu, kterou můžete vidět **na obrázku 13**. Tato třída se stará o prvotní kontrolu souboru, její atributy reprezentují plnou cestu ke vstupnímu souboru Enum s podporovanými datovými typy a samotný vstupní soubor. Při vytváření instance této třídy je potřeba znát cestu ke vstupnímu souboru, která se předá konstruktoru. Ten poté nastaví svůj atribut „fileName“ a podle něj vytvoří vstupní soubor, který poslouží ke kontrole. Všechny atributy pro cestu k souboru a samotný vstupní soubor jsou nastaveny jako konstanta.

Jediná veřejná metoda „checkFile“ slouží ke kompletní kontrole vstupního souboru. Jejím návratovým typem je jedna z hodnot již zmíněného Enumu „FileTypes“. V prvé řadě tato metoda kontroluje, zda není cesta ke vstupnímu souboru nulová (prázdný řetězec). Pokud ano, metoda se ukončí a vrátí hodnotu „isNull“, pokud ne, metoda pokračuje dále ke kontrole samotného vstupního souboru.



Obrázek 13 Diagram třídy FileCheck

Zde se kontroluje, zda zadaná cesta vede k souboru a jestli je tento soubor nenulový. Pokud jsou tyto podmínky splněny, metoda určí příslušný datový typ vstupního formátu, který zjistí z konstanty „fileName“ pomocí regulárního výrazu. Pokud nebude mít vstupní soubor koncovku podporovaného souboru, metoda nastaví vstupní formát na hodnotu „notSupported“. V případě, že cesta nevede k souboru, nebo soubor je nulový, vstupní formát se nastaví na hodnotu „isNull“. Na konci metody se vrátí zjištěný typ vstupního souboru.

Metoda „checkCoding“, která je použita v kontrolní třídě, slouží ke zjištění kódování vstupního souboru. To je potřeba znát především při použití formátu CSV, kde není kódování dáno žádnou normou. Metoda využívá externí knihovnu CharsetDetector a CharsetMatch. Tyto knihovny určí, které kódování nejlépe odpovídá vstupnímu souboru a vrátí jej v podobě řetězce.

Po ukončení metody „checkFile“ postup programu vrátí zpět do řídící třídy, kde se vyhodnotí výsledky výstupu ukončené metody a podle toho postupuje dále. Pokud má výsledek hodnotu „isNull“, „notSupported“ nebo se výsledek neshoduje s hodnotou zadanou uživatelem, pak se program ukončí a vrátí specifikovanou chybovou hlášku. V případě, je výsledek korektní, metoda pokračuje voláním privátní metody „convertInputToJson“.

Tato metoda má za úkol převést vstupní formát na mezi-formát (JSON). K tomu potřebuje několik parametrů – Vstupní datový typ, který byl zjištěn v předchozích krocích, kódování souboru (zjištěno v metodě „checkCoding“) a plnou cestu ke vstupnímu souboru. Tato funkce má návratový typ String, který je deklarován na začátku funkce a později se do něj uloží zpráva o převodu vstupního souboru.

Pomocí přepínače Switch a vstupního parametru „fileType“ se spustí jeden z převodníku vstupního souboru na formát JSON. Jednotlivé převodníky jsou popsány v níže uvedené kapitole 5.1.2 Jednotlivé převodníky vstupu.

Funkce „runAnalyse“ a se používá ke spuštění analýzy, pouze v případě, kdy je tato analýza potřeba. Především tedy při výstupu typu SQL. Metoda „provideAnalyzedItems“ se používá, pro výpis analyzovaných prvků uživateli. Ten si pak může prohlédnout jednotlivé sloupce, délku (v případě řetězce), datový typ a případně jej změnit na jiný.

5.1.1 JSONObject a JSONArray

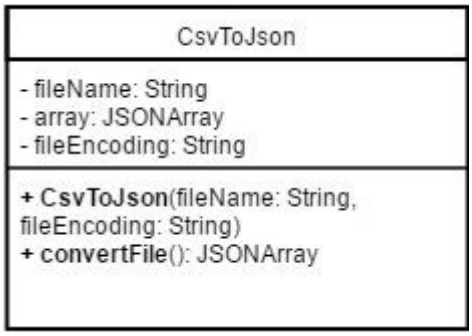
Pro převod vstupního souboru na formát JSON jsem implementoval převodník ke každému vstupnímu formátu. V každém převodníku se používá instance objektu JSONObject nebo JSONArray, které jsou součástí externí knihovny org.json. Tato knihovna slouží k reprezentaci formátu typu JSON v operační paměti. Krom jiného dokáže načíst a převést text ve formátu JSON ze souboru, nebo řetězce. Knihovna má metodu pro převod CSV textu na formát JSON a také pro převod z a do XML formátu.

Jak již napovídají názvy tříd, JSONObject reprezentuje objekt typu JSON, tedy pár klíč / hodnota. Ty jsou uloženy do hashovací tabulky, která si neuchovává pořadí vkládaných prvků. Klíč může být libovolný řetězec, hodnota pak může nabývat různých formátů od běžného řetězce, přes další JSONObject až po JSONArray. Třída má v sobě integrováno spoustu metod pro práci s JSON formátem, jak pro čtení, tak pro zápis. Třída JSONArray pak reprezentuje pole prvků (hodnot), které mohou být různých typů stejně jako JSONObject. Detailnější informace o JSON objektu a JSON Array jsou uvedeny v kapitole 2.4 JSON.

5.1.2 Jednotlivé převodníky vstupu

CSV

Níže uvedený obrázek reprezentuje třídu převodníku vstupního formátu na mezi-formát JSON. Třída má tři proměnné, první z nich reprezentuje plnou cestu ke vstupnímu souboru typu CSV. Druhá proměnná je typu JSONArray a poslouží k uložení JSON do paměti a předání k dalšímu zpracování. Poslední proměnná určuje kódování vstupního souboru. Cesta i kódování souboru jsou konstanty, které se nastavují v konstruktoru při vytváření objektu.

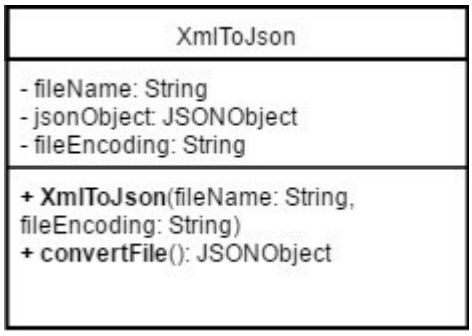


Obrázek 14 Třída převodníku CSV na JSON

Veřejná metoda „convertFile“ zajišťuje převod vstupního CSV souboru na objekt typu JSONArray. Pro tento převod je nejprve nutné načtení celého obsahu CSV souboru do řetězce typu String. Načítání se provádí řádek po řádku v cyklu While. Kompletní řetězec s načtenými daty se potom použije jako vstup funkce knihovny org.json - CDL.toJSONArray. Tato funkce převede daný odřádkovaný řetězec na objekt typu JSONArray. První řádek v souboru se bere jako hlavička souboru, hodnoty tohoto řádku budou brány jako klíče k dalším hodnotám. Pří úspěšném převodu tato funkce vrátí objekt typu JSONArray.

XML

Obrázek 16 reprezentuje třídu pro převod vstupního XML na mezi-formát JSON. Stejně jako předchozí třída, i tato má tři proměnné, první z nich reprezentuje plnou cestu ke vstupnímu souboru typu XML. Druhá proměnná je typu JSONObject a poslouží k uložení JSON do paměti a předání k dalšímu zpracování. Poslední proměnná určuje kódování vstupního souboru. Cesta i kódování souboru jsou konstanty, které se nastavují v konstruktoru při vytváření objektu.

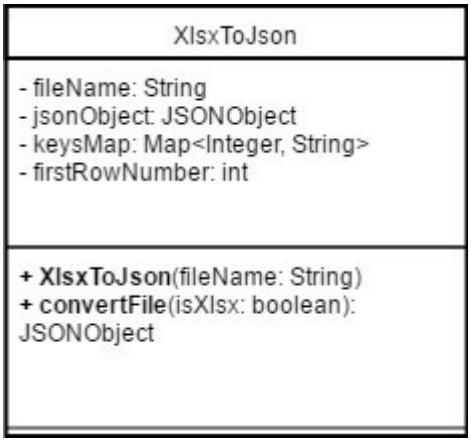


Obrázek 15 Třída převodníku XML na JSON

Veřejná metoda „convertFile“ funguje v podstatě úplně stejně jako v předchozím případě. Využívá se v ní pouze jiná funkce knihovny org.json, a to konktrétně XML.toJSONObject. Tato funkce převede vstupní XML řetězec na formát JSON, který se dále uloží do proměnné „jsonObject“, která je zároveň výstupem funkce. Při převodu na formát JSON se zachovává zanoření prvků čili i struktura dat.

XLS a XLSX

Pro implementaci převodníku vstupního i výstupního formátu XLSX (XLS) jsem použil knihovnu Apache POI, která umožňuje široké možnosti použití. Tato knihovna nabízí možnost vytvářet, upravovat a číst soubory ve formátu XLS i novém formátu XLSX. Neomezuje se však jen na ně, umožňuje práci s celou sadou dokumentů společnosti Microsoft. V této práci se však budu zabývat pouze funkcemi souvisejícími s výše uvedenými formáty aplikace Excel.



Obrázek 16 Třída převodníku Xlsx (Xls) na JSON

Výše uvedený obrázek třídy „XlsxToJson“ zobrazuje všechny funkce a proměnné. Stejně jako obě předchozí třídy i tato má proměnnou reprezentující cestu k souboru a proměnou pro uložení výsledného objektu ve formátu JSON. Proměnná „keysMap“ slouží průběžnému ukládání dat jednotlivých hodnot a klíčů do paměti předtím, než se vloží do JSON objektu. Poslední proměnná slouží k určení prvního řádku, na kterém se nachází nějaký záznam. Vzhledem k tomu, že před prvním záznamem může být neurčité množství prázdných řádků, je potřeba ošetřit, aby se řádek s indexem 0 nebral automaticky jako první.

Třída obsahuje stejně jako ostatní převodníky metodu, která tento převod zajistí. Tato metoda vyžaduje pro správnou funkčnost jeden argument typu boolean, ten určí, zda je vstupní soubor typu XLSX, nebo XLS. Ačkoli je práce s oběma soubory zcela totožná (ve smyslu průchodu souborem), musíme je rozlišit z důvodu různého zápisu na disku (rozdíly viz kapitola 3.2).

Dále si postupně rozebereme celou metodu pro převod. Prvním krokem metody je načtení vstupního souboru ze zadaného řetězce cesty. Ten se načte do paměti pomocí instance třídy „Workbook“, jež je součástí knihovny Apache Poi. Tato třída reprezentuje pracovní sešit, stejný jako je v aplikaci Microsoft Excel. Po načtení souboru do sešitu přichází na řadu rozhodnutí, zda se jedná o pracovní sešit typu XLS nebo XLSX. Pro rozhodnutí použijeme argument metody, podle kterého přetypuje pracovní sešit. K přetypování využijeme instanci třídy XSSFFormulaEvaluator respektive HSSFFormulaEvaluator (součást Apache Poi). Po přetypování získáme pracovní sešit ve formátu XLSX (XSSFWorkbook), respektive ve formátu XLS (HSSFWorkbook). Poté už tyto různé vstupní formáty nerozlišujeme.

Při dalším pokračování si metoda zjistí, kolik je v pracovním sešitu obsaženo listů (Sheets). Tak, abychom mohli projít a převést všechny do jednoho JSON objektu. Poté si metoda vytáhne Iterátor listů a začne postupně procházet všechny listy pracovního sešitu. Do dočasné proměnné si uloží aktuálně zpracovávaný list a zjistí si pozici prvního řádku obsahující nějaký záznam. List (Sheets) může a vždy obsahuje několik řádků, proto je musíme projít všechny stejným způsobem jako předtím listy. Metoda si načte Iterátor řádků a postupně je začne procházet cyklem. Poslední třetí průchod se bude týkat buněk (Cell), ty projdeme stejným způsobem jako předchozí listy a řádky. První podmínka v cyklu průchodu filtruje ty buňky, které se nacházejí na prvním řádku. Buňky budou totiž označeny jako klíče k hodnotám v následujících řádcích.

Dalším krokem je jednotlivé buňky uložit ve správném pořadí, ve smyslu hodnota ke správnému klíči, do výsledného JSON objektu. V tom nám pomůže proměnná „keysMap“, do které si uložíme klíče spolu

s indexem sloupce, ve kterém se tento klíč nachází. Předtím než tak uděláme, však musíme zkontrolovat typ buňky. I když si ukládáme jediný datový typ – řetězec, buňka může nabýt hned několika. Konkrétně to je řetězec, číslo, rovnice, chyba, boolean a chybějící hodnota. Vzhledem k tomu, že může nastat případ prázdné buňky hned v prvním řádku, nebo jakémkoli jiném řádku, je potřeba zkontrolovat každou buňku, jestli není náhodou nulová.

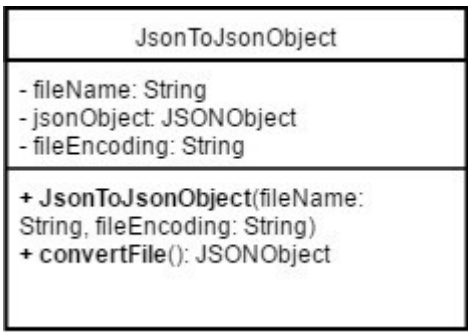
Pokud se jedná o první řádek, zde rozlišujeme pouze dva typy buněk – řetězec a prázdné buňky, ostatní typy buněk zde přenastavíme na typ řetězec, pro účel parseru jsou totiž nepodstatné. V případě, že je buňka typu řetězec, uložíme ji spolu s indexem (klíč) do proměnné „keysMap“. Pokud daná buňka chybí, do proměnné uložíme index a jako hodnotu programátorem zadaný řetězec označený indexem buňky. Díky tomuto postupu nám nebudou chybět žádné klíče pro sloupce v sešitu.

Pokud se nejedná o první řádek, postupuje se v podstatě stejně, s tím rozdílem, že data již ukládáme do výsledného JSON objektu. Klíč pro danou hodnotu buňky si vždy vytáhneme z proměnné „keysMap“ podle indexu aktuální buňky. Prázdný typ buňky nás v tomto případě již nezajímá, jelikož každý sloupec nemusí být povinný údaj. Naopak nás zde zajímá typ číslo, do kterého se v Excelu ukládá datum. Tento typ je třeba převést na text tak, abychom s ním mohli v další fázi programu pracovat. K tomu se použije třída „FormulaEvaluator“ (součást Apache Poi).

Poté, co funkce projde všechny buňky, řádky i sešity, vrátí výsledný objekt typu JSONObject.

JSON

Třída „JsonToJsonObject“ neslouží k převodu, ale spíš k načtení souboru ve formátu JSON do paměti (JSON objektu) pro další zpracování programem. Proměnné této třídy jsou klasicky cesta k souboru, kódování (nastavitelné v konstruktoru) a objekt typu JSONObject, který se vrací v metodě „convertFile“.

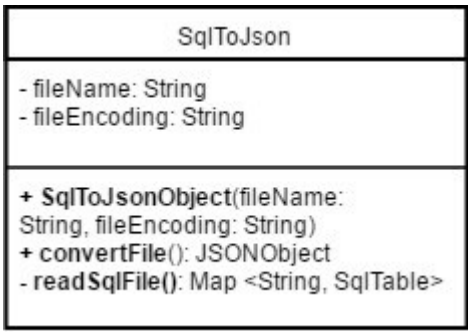


Obrázek 17 Třída převodníku JSON souboru na JSON

Metoda pro převod (načtení) JSON souboru využívá funkci z knihovny Apache Commons, konkrétně IOUtils.toString(). Ta převede vstupní stream na textový řetězec, který stačí vložit do nově vytvořeného JSON objektu. Probíhá zde i oprava v případě chybějící syntaxe (obalení JSON).

MySQL dump

Posledním podporovaným vstupním formátem je formát SQL, konkrétně se jedná o databázový výstup – MySQL dump. Proměnné třídy jsou cesta k souboru a jeho kódování, obě jsou konstanty a nastavují se v konstruktoru. Třída má jednu veřejnou funkci, která je volána z kontrolní třídy a zajistí převod vstupu na výstup v podobě JSONObject objektu.



Obrázek 18 Třída převodníku SQL dump souboru na JSON

Privátní metoda „readSqlFile“ se stará o samotné načtení SQL souboru. To se provádí řádek po řádku v cyklu While. Každý řádek pak prochází kontrolou pomocí regulárních výrazů. Regulární výrazy hledají klíčové řetězce syntaxe použité v MySql dump souboru a podle toho načítají data do paměti. Funkcionalita regulárních výrazu je zabudovaná přímo v interní třídě *String* jazyka Java. Lze tedy jednoduše určit, zda konkrétní řetězec obsahuje daný vzor. Příklad použití regulárních výrazů lze vidět v níže uvedeném kódu. Zde si do řetězce ukládám řádek po řádku data ze vstupního souboru a následující podmínce hledám konkrétní vzor „.*INSERT INTO.*“ .*, nebo „insert into.*“. Spojení symbolu tečky a hvězdičky udává, že před konkrétním řetězcem může být obsažen jakýkoli jiný obsah. Hvězdička následovaná tečkou určuje to samé, ale za řetězcem. Rovné lomítko mezi těmito dvěma vzory určuje, že může platit buď jeden, nebo druhý vzor, aby byla podmínka splněna.

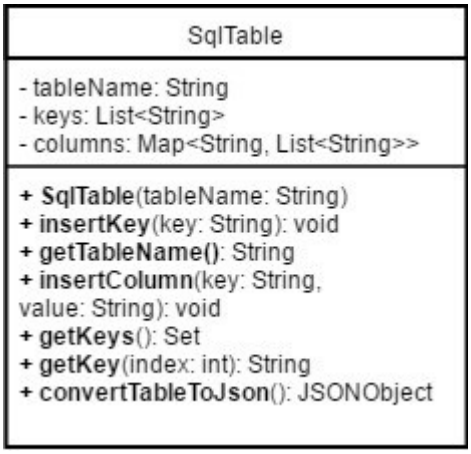
```
while (scanner.hasNextLine()) {
     String line = scanner.nextLine();

     if (line.matches(".*INSERT INTO.*|.*insert  into.*")){
         Pattern pattern = Pattern.compile("`(.*?)`");
         Matcher match = pattern.matcher(line);
         String tableName;

         if(match.find()){
```

Pro komplexnější využití v další implementaci této metody bylo krom zabudované funkce Stringu, nutno použít interní třídy Pattern a Matcher. Obě třídy slouží pro práci s regulárními výrazy. Třída Pattern slouží k vložení regulárního výrazu, který se poté vyhledává v řetězci a výsledek vyhledávání se ukládá právě do instance třídy Matcher. Výsledků vyhledávání vzoru může být několik, přistupuje se k nim prostřednictvím Matcheru. Ve výše uvedeném příkladu se v řádku vyhledává vzor, který je ve stejném řádku jako předchozí vyhledávaný výraz (Insert Into). Tento výraz musí být zleva i zprava obklopen určeným typem uvozovek, ve kterých může být prakticky cokoliv. Logicky z této posloupnosti vyjde název tabulky.

S použitím regulárních výrazů je funkce „readSqlFile“ schopná vyfiltrovat názvy všech tabulek, do kterých se vkládají nějaké hodnoty. Dále je funkce schopna vyfiltrovat jména jednotlivých sloupců a všech vkládaných hodnot pro jednotlivé sloupce. Tato data funkce ukládá do mapy typu LinkedHashMap, která reprezentuje seznam všech tabulek (databázi). Klíčem této proměnné je jméno tabulky a hodnota je instance třídy SqlTable.



Obrázek 19 Diagram třídy SqlTable

Tato třída slouží k ukládání dat určité tabulky, jedna instance této třídy reprezentuje právě jednu tabulku. Proměnná „tableName“ specifikuje její název a „columns“ pak její sloupce – jméno sloupce a hodnoty v podobě pole. Proměnná „keys“ je potom seznam všech názvů sloupců, slouží k jednodušší manipulaci. Jednotlivé metody této třídy slouží k získávání či vkládání hodnot, vyjma veřejné metody „convertTableToJson“. Tato metoda převede proměnnou „columns“ do podoby JSONObjektu pomocí metody „putOnce“ stejnojmenné třídy. Díky této funkci zůstanou prvky v JSON objektu ve stejném pořadí, jako byli v proměnné „columns“.

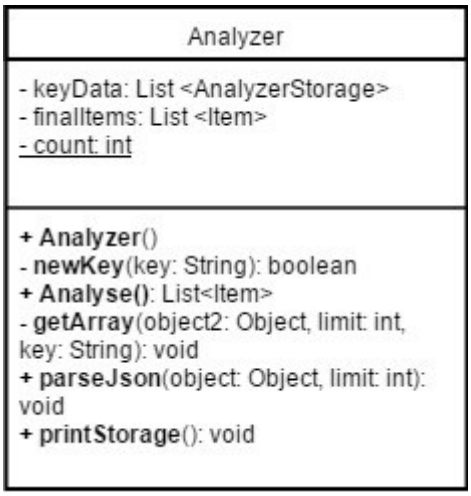
Jednotlivé instance třídy „SqlTable“ se vytvářejí a vždy při nalezení vzoru, uvedeného v příkladu výše. Poté už se do tabulky postupně ukládají nové sloupce a k nim příslušné hodnoty. Součástí vkládací metody je i kontrola, zda vkládaný název sloupce není nulový, nebo jestli už neexistuje.

Po úspěšném průchodu funkce „readSqlFile“ jsou všechna data načtena v proměnné typu Map <String, SqlTable>, veřejné funkce „convertFile“ (třídy SqlToJson). Metoda dále, nad každým prvkem této mapy, zavolá funkci „convertTableToJson“ a všechny sestavené JSON objekty pak obalí do jednoho konečného objektu, který vrátí zpět do řídící třídy.

Úskalí této třídy spočívá v nutnosti přesně formátovaného vstupního souboru SQL, regulární výrazy totiž netolerují žádnou odchylku od zadaného vzoru. Proto je nutné dodržovat určitou syntaxi vstupu. Příklad vstupního mySQL dump souboru (se správnou syntaxí) je obsažen v příloze této práce pod názvem SQL_dump.slq

5.2 Analýza dat

Nezbytnou součástí této práce je analýza vstupních dat. Ta je nezbytná při vytváření SQL výstupního formátu, zde totiž potřebujeme znát datové typy jednotlivých sloupců tabulky. Pro jejich určení využívám třídu s názvem „Analyzer“. Ta je zobrazena na níže uvedeném obrázku. Třída má dvě proměnné pro uložení dat pro analýzu z JSON objektu (keyData), který bude analyzovat a pro uložení výsledku této analýzy (finalItems). Tyto dvě proměnné jsou datového typu List a jejich vnitřním datovým typem je třída vytvořena na míru této aplikace. Obě třídy a jejich význam bude popsán dále v textu.



Obrázek 20 Diagram třídy Analyzer

Před spuštěním samotné analýzy dat, je potřeba v řídicí třídě vytvořit instanci třídy „Analyzer“ a naplnit ji daty z mezi-formátu. To uděláme tak, že spustíme veřejnou metodu „parseJson“, ta nemá za úkol nic jiného, než postupně projít celý JSON a vytáhnout z něj všechny klíče a určený počet hodnot těchto klíčů. Počet je reprezentován atributem „limit“, který zadá uživatel aplikace, případně se použije defaultní hodnota. Tento počet určí, nad kolika prvky každého klíče se má spouštět analýza. Platí zde přímá úměra, čím víc prvků se bude analyzovat, tím přesnější bude výsledek. Ovšem také platí, čím více prvků pro analýzu, tím delší čas k jejímu provedení.

Vstupem metody „parseJson“ je, krom čísla, obecný objekt typu Object. Ten reprezentuje vstupní data (mezi-formát), z kterého se vytahují jednotlivé klíče a hodnoty. Obecný objekt je zvolen z toho důvodu, že JSON může ze své definice obsahovat buď JSON objekt, nebo JSON pole (popsáno v kapitole JSON), v mojí aplikaci může JSON obsahovat ještě instanci typu LinkedHashMap, která se vkládá do objektu v případě vstupu z SQL dump souboru. Tato instance je zde kvůli zachování pořadí jednotlivých prvků. Samotný JSONObject používá pro uchování dat typ HashMap, která si pořadí nezachovává.

Metoda tedy logicky začíná kontrolou objektu, kde se zjišťuje, zde je objekt instance JSONObject, nebo JSONArray. SQL vstup je obalen v JSON objektu, takže splní první podmínku. Pokud je objekt typu JSONObject, metoda si z něj vytáhne všechny klíče, pomocí Iterátoru je začne projíždět. Přitom si přidá nový prvek do proměnné „keyData“ pro uložení klíče a hodnot pro pozdější analýzu.

Při průchodu klíči se kontroluje, zda hodnota pod konkrétním klíčem není typu JSONObject, JSONArray, nebo LinkedHashMap. Pokud je typu pole, metoda zavolá privátní metodu „getArray“,

předá do ní klíč, limit pro analýzu a objekt instance JSONArray, který byl vyjmut ze vstupního objektu aktuálním klíčem. Tato metoda bude popsána v následujících odstavcích.

Pokud je hodnota typu JSONObject, zavolá se znovu metoda „parseJson“ s limitem a pro analýzu a objekt instance JSONObject, který byl vyjmut ze vstupního objektu aktuálním klíčem. Pokud je hodnota typu LinkedHashMap, pak ji převedu na tento typ (přetypuji) a zpracuji jednotlivé klíče a hodnoty uložené v této hashovací tabulce. Všechny klíče v této tabulce uložím do proměnné „keyData“ a ke každému takovému klíči přidám určený počet hodnot.

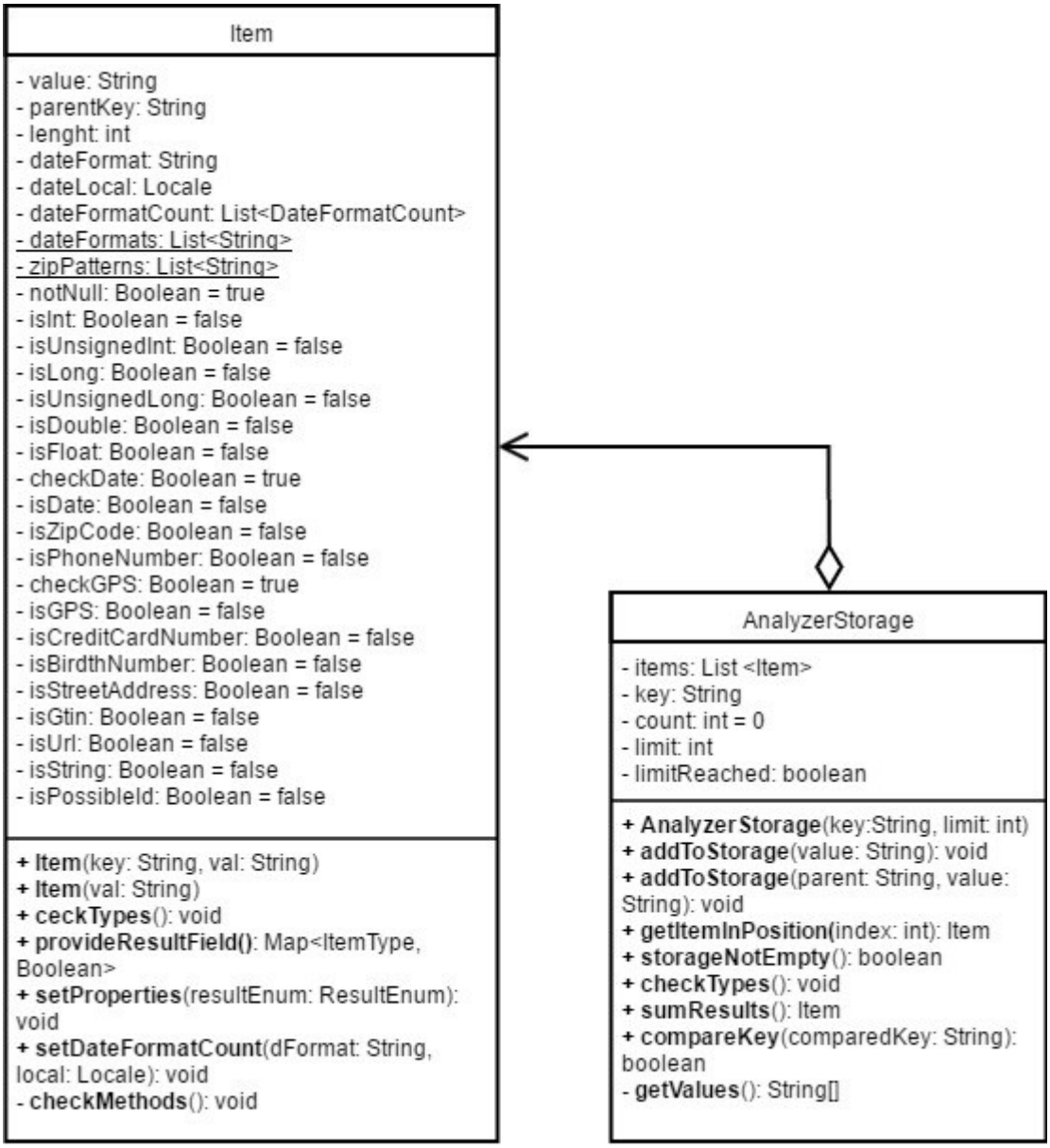
Pokud není hodnota ani jednoho typu z uvedených typů, znamená to, že v sobě neskrývá žádná strukturovaná data a jedná se tedy o hodnotu aktuálního klíče. Tato hodnota vloží do proměnné „keyData“, v případě, že je již k danému klíčí přidán dostatečný počet hodnot, pak se tato hodnota přeskočí. V případě že klíč ještě nebyl vložen do proměnné, pak se vytvoří nový a rovnou se do něj vloží hodnota. Hodnota může nabývat různých typů od řetězce, přes číslo, až po datum. Program si ji však ukládá jako řetězec, který bude vzápětí analyzovat.

Funkce „getArray“ funguje skoro stejně jako ta předešlá. Na jejím začátku se obecný objekt přetypuje na typ „JSONArray“ (nic jiného se do této funkce nedostane), poté si velikost pole a projíždí prvek po prvku. Přitom zjišťuje typ aktuálního prvku, který může být buďto JSONObject nebo jednoduchá hodnota. V případě objektu JSON se opět zavolá metoda „parseJson“ do které se pošle zjištěný objekt. V případě jednoduché hodnoty se tato hodnota vloží do proměnné „keyData“ pod příslušným klíčem, který byl metodě předán jako parametr. V případě, že se jedná o nový klíč, se v této proměnné vytvoří nová položka.

Na níže uvedeném Obrázek 21 lze vidět diagram třídy „Item“ a „AnalyzerStorage“ obě třídy se používají k uchování dat při analýze. Třída Item představuje základní prvek, do kterého se ukládají surové informace – klíč (rodič) a hodnota, obě proměnné jsou typu řetězec. Třída má dále proměnou „lenght“, ta reprezentuje délku řetězce. Všechny tři hodnoty se nastavují v konstruktoru, ať už v tom s hodnotou i klíčem, nebo jenom s hodnotou. Další proměnná dateFormat se používá pouze v přídě, že je hodnota typu datum. V takovém případě tato proměnná po analýze obsahuje konečný formát data. Proměnná „dateLocal“ je instancí třídy „Local“, používá se pouze při kontrole data a určuje, zda je datum v českém nebo anglickém formátu (lze nastavit i na jiné).

Proměnná dateFormatCount představuje pole různých datumových formátu, které se používá při analýze dat. Do toho pole se ukládá právě určitý datumový formát (enum „DateFormatCount“), který počítá, kolikrát se tento formát vyskytl pro daný klíč. Do finálního výsledku se vybere ten, který se vyskytl nejčastěji. Statická proměnná „dateFormats“ představuje seznam všech podporovaných formátů dat. Obdobná statická proměnná je i „zipPatterns“, která reprezentuje různé formáty PSČ adres.

Všechny ostatní třídní proměnné jsou typu Boolean a označují, zda je konkrétní hodnota daný datový typ nebo ne. K potřebným proměnným jsou vytvořeny metody pro nastavení a získání jejich hodnot. Ke všem proměnným typu Boolean jsou vytvořeny metody, které je nastavují a určují tak, jaký datový typ hodnota představuje. Tyto metody nejsou v diagramu obsaženy z důvodu redukce jeho velikosti. V dalších odstavcích stručně popíši vybrané metody pro analýzu typu dat.



Obrázek 21 Diagram tříd Item a AnalyzerStorage

Metody pro určení číselných datových typů využívají zabudované funkce jazyku Java, konkrétně jeho metod pro přetypování objektu na číselný objekt. Přitom využívám bloku try-catch, kdy zkouším převést daný textový řetězen na určené číslo. V případě, že se převod v bloku try podaří, nastavím příslušnou proměnnou na hodnotu true. Pokud se převod nezdaří, metoda skočí do bloku catch a příslušnou proměnou nastavím na hodnotu false. Tento postup lze vidět na příkladu metody (checkInt) níže.

```
private void chechkInt(){
    try{
        Integer.parseInt(value);
        this.isInt = true;
        this.checkDate = false;
        this.checkGPS = false;
    }catch(NumberFormatException e){
    }
}
```

Uvedená metoda kontroluje, zda je hodnota číslo (int) o délce 10 číslic a se znaménkem. Všechny metody, vyjma metody pro určení data a GPS souřadnic, v případě pozitivního bloku try nastaví proměnnou „checkGPS“ a „checkDate“ na hodnotu *false*, toto zamezí zbytečnému spuštění těchto dvou metod, které při analýze zabírají nejvíce času.

Metoda pro určení data má jako jediný atribut řetězec označující jazyk vstupu, lépe řečeno tag tohoto jazyka. Metoda je spuštěna nejprve s jazykovým tagem „en“, ten označuje americký typ dat, v případě, že datum nebude nalezeno, metoda se spustí ještě jednou s tagem „cs-CZ“ pro české typy dat. Díky tomuto nastavení lze rozpoznat datum skládající se ze slov dnů či měsíců, a to jak v anglickém, tak českém jazyce.

Kontrola různých datumových formátů se provádí pomocí zabudovaných tříd v Javě – „DateTimeFormatter“ a „LocalDate“. První vytvoří vzor pomocí jednoho z datumových formátů a již zmíněného jazykového tagu. Tento vzor se poté kontroluje pomocí druhé třídy proti vstupní hodnotě, což je proměnná „value“. Všechny různé formáty dat jsou uloženy ve statické proměnné „dateFormats“ příklad formátu je - "**eeee, MMMM d, yyyy**" ten reprezentuje tento vstup: „*Monday, June 30, 2014*“ v češtině: „*Pondělí, června 30, 2014*“. V případě nalezení formátu data se příslušná proměnná nastaví na hodnotu *true*, do proměnné „dateFormat“ se uloží formát data a do proměnné „dateLocal“ se uloží jazykový tag pro formát data.

Zbylé funkce pro kontrolu pracují s regulárními výrazy, podobně jako tomu bylo u funkce pro převod vstupního SQL souboru na mezi-formát. Používá se zde funkce zabudovaná v třídě String, ta porovnává vložený regulární výraz s hodnotou proměnné. V níže uvedeném příkladu se kontrolují dva vzory GPS souřadnic, konkrétně *50.081819602*, *14.44152832, 50° 4' 54.5505674" N* nebo *12°30'23.256547\"E*. Zbylé metody se liší prakticky jen v použitém regulárním výrazu.

```
    private void checkGps(){
        if(value.matches("^[-+]?([1-8]?\\d(\\.\\d+)?|90(\\.0+)?),\\s*[-
+]?(180(\\.0+)?|((1[0-7]\\d)|([1-9]?\\d))(\\.\\d+)?)$")){
            this.isGPS = true;
            this.isString = false;
        }
        else if(value.matches("[0-9]{1,2}[:|°]? ?[0-9]{1,2}[:|']? ?(?:\\b[0-
9]+(?:\\.[0-9]*)?|\\.[0-9]+\\b)\"? ?((N)|(S)|(E)|(W))?")){
            this.isGPS = true;
            this.isString = false;
        }
        else{
            this.isString = true;
        }
    }
```

Pro spuštění všech metod analýzy se používá veřejná metoda „checkType“. Metoda „provideResultField“ vytvoří proměnou typu EnumMap, do které uloží výsledky jednotlivých kontrol spolu s klíčem typu ItemType.

Funkce „setProperties“ se vstupním parametrem typu „ResultEnum“ se používá k přenášení výsledných dat (sumarizaci) ve třídě „AnalyzerStorage“. Metoda porovná vstupní proměnnou a porovná ji se všemi položkami výčtu typu „ResultEnum“. Pokud se porovnávání shoduje, pak nastaví příslušnou proměnou objektu typu „Item“ na **true**, objekt tedy bude mít pouze jednu proměnou reprezentující datový typ s hodnotou true. Metoda „setDateFormatCount“ se stará o práci s různými formáty dat, metoda postupně vkládá nalezené formáty při analýze.

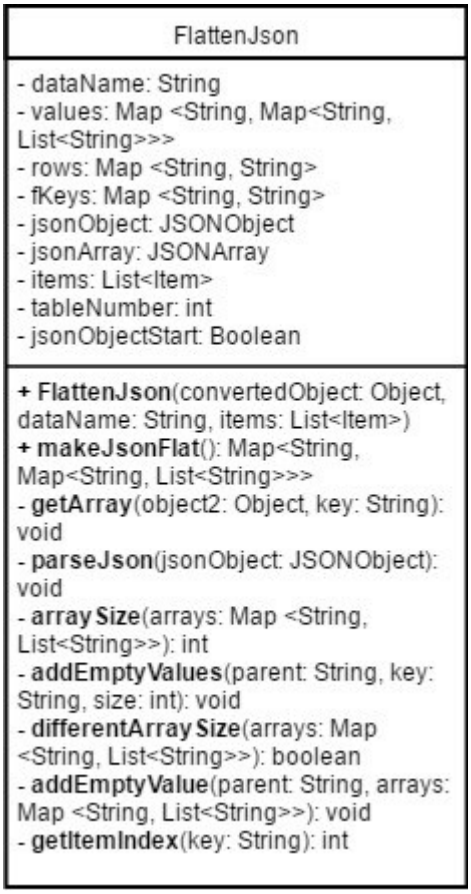
Na výše uvedeném Obrázek 21 je uveden i diagram třídy „AnalyzerStorage“. Do této metody se ukládají data, nad kterými se spustí analýza, spouští se z ní metody třídy „Item“, funkcí „checkTypes“ – spustí kontrolu nad všemi prvky své proměnné „items“. Dále tyto výsledky kompletuje do podoby jediného objektu typu „Item“. Třída obsahuje i funkci pro zjištění a zaokrouhlení délky řetězce, která je potřebná při vytváření databázového výstupního formátu.

Analýza vstupu se spouští v řídicí třídě, a to prostřednictvím instance třídy „Analyzer“ a její funkce „analyse“. Tato metoda pouze spouští jednotlivé výše uvedené metody a ukládá si výsledky analýzy do pole konečných výsledku. V poli tedy nalezneme tolik prvků, kolik je v JSON objektu klíčů. Výsledné pole prvků pak tato metoda předá do řídící třídy, která s ním dále pracuje. Metoda „printStorage“ slouží pouze ke kontrole obsahu, na konzoli vypíše celý obsah proměnné „keyData“ pole.

5.3 Převod dat na výstupní formát

Posledním krokem této aplikace je převést mezi-formát JSON na uživatelem zadaný výstupní formát pomocí dat, které jsme získali v předchozí analýze. K převodu se používá metoda třídy „ParseController“ s názvem „convertToOutput“. Vstupem této metody je – **Enum** (s datovým typem, na který chceme soubor převést), **kódování souboru** (zjistili v předchozích krocích), **cesta** (kde se má výstupní soubor vytvořit), **název výstupního souboru** (včetně příslušné koncovky), **objekt** (obsahující mezi-formát JSON), dále **analyzovaná data** (z předešlé analýzy) a **typ databáze** (pro kterou se má vytvořit výstupní soubor. Pouze v případě výstupu SQL).

Metoda pokračuje dále stejným způsobem jako při určování vstupu. Přepínač Switch rozhodne o tom, jaký je výstupní typ a podle toho zavolá příslušné metody k transformaci a uložení výstupního souboru. Při převodu z mezi-formátu na výstupní formáty se v některých případech využívá mnou vytvořená třída „FlattenJson“. Ta zajišťuje zploštění formátu JSON a zruší jeho zanoření, které není možné reprezentovat 2D formáty, jako je například CSV, nebo XLSX.



Obrázek 22 Diagram třídy FlattenJson

Konstruktor této třídy vyžaduje dva parametry, a to je objekt v mezi-formátu JSON, který jsme dostali v přechozích krocích aplikace a pole s analyzovanými prvky. Kontrola, zda je objekt typu JSONObject, nebo JSONArray, se provádí již v konstruktoru a začátek průchodu se určuje tímto objektem pomocí proměnné „jsonObjectStarts“, který se taktéž nastavuje v konstruktoru.

Jediná veřejná metoda této třídy „makeJsonFlat“ se stará o celý proces „zploštění“. Jejím návratovým typem je pak výsledný 2D objekt typu LinkedHashMap, ten vypadá v podstatě stejně jako databáze s tabulkami, sloupci a řádky. Metoda začíná podmínkou, která rozhoduje, zda se začne průchod metodou „parseJson“ nebo „getArray“. Obě metody fungují obdobně jako ve třídě Analyzer, která je popsána v předchozí kapitole. Největší rozdíl mezi implementacemi těchto dvou párů metod je v tom, že metody z třídy FlattenJson se snaží krom klíče a hodnoty zjistit, zda není daný prvek zanořen. Takové zanoření by představovalo referenci na cizí klíč. Tyto reference se ukládají do proměnné „fKeys“ typu Map<String, String>. Dvojice klíč / hodnota zde reprezentuje vztah rodič / potomek.

Metoda se navíc stará o to, aby byla nalezena prázdná místa, a do nich doplněny nulové hodnoty (prázdné řetězce). JSON formát totiž v případě nulové hodnoty klíče tento klíč ve své struktuře neuvede. Tento fakt by bez ošetření mohl vést k tomu, že by byly záznamy uvedeny v nesprávném pořadí. Pro lepší pochopení situace uvedu příklad takového JSON souboru.

```
{
    "flight": [
        {
          "foo":"bar",
          "coolness":"2.0",
          "altitude":"32434",
          "pilot": {
              "firstName":"Buzz",
              "lastName":"Aldrin"
          },
          "mission":"apollo 11"
        },
        {
          "foo":"bar2",
          "coolness":"3.0",
          "pilot": {
              "firstName":"Buzz",
              "lastName":"Maldrin",
              "id":"2"
          },
          "mission":"apollo 12"
        },
        {
          "foo":"bar3",
          "coolness":"4.0",
          "altitude":"41150",
          "pilot": {
              "firstName":"Juzz",
              "lastName":"Kaldrin",
              "id":"3"
          },
          "mission":"apollo 13"
        }
    ]
}
```

Tento příklad má několik různě zanořených dvojic klíč: hodnota. Nás bude zajímat dvojice s klíčem „id“, ta je totiž v prvním objektu „pilot“ nulová. Tím pádem by se při převodu stalo to, že by se klíč „id“ vytvořil až při druhém průchodu pilotem a hned by do sebe vložil hodnotu „2“. Při vytváření výstupu se však data čtou postupně, jelikož jsou uloženy v poli, takže by se hodnota „2“ nepřidělila k druhému pilotovy, ale rovnou k prvnímu. Nulová hodnota by byla vložena až na konec pole, takže by byla vložena

ke 3. pilotovi. Správné posunutí a umístění nulových hodnot řeší metody „arraySize“, „addEmptyValues“, „differentArraySize“, „addEmptyValue“ a „getItemIndex“. Poté, co se JSON objekt správně převede do 2D podoby, metoda vrátí již zmíněné asociativní pole zpět kontrolní třídě pro další zpracování. Ta pokračuje vytvořením instance požadovaného převodníku na výstupní formát. Každý z těchto převodníků vyžaduje cestu pro uložení výstupního souboru a jeho název. Dalším parametrem, který převodník vyžaduje je buď objekt typu JSON, nebo jeho zploštělá 2D podoba.

CSV

Metoda pro převod na výstupní formát CSV vyžaduje plochá data, strukturovaná data totiž neumí interpretovat. Tento převodník má navíc ještě jeden parametr konstruktoru typu Boolean„intoOneFile“. Tento parametr určuje, zda budou všechna data uložena do jediného CSV souboru, nebo jestli se budou ukládat do více souborů, tak aby mohly svým způsobem interpretovat strukturovaná data. Samotný převod do CSV formátu již není nijak složitý. Asociativní pole se začne postupně procházet, vždy se projdou všechny „tabulky“ pak všechny „sloupce“ a nakonec všechny „řádky“ ve kterých jsou uloženy jednotlivé hodnoty.

Zde však musím prohodit sloupce s řádky, protože v CSV jsou data sice v po sobě jdoucích řádcích, ale hodnoty jednoho klíče jsou uloženy v jednom sloupci. Jednotlivé „tabulky“ jsou pak uvedeny v samostatných souborech, nebo v jednom komplexním, podle zadání uživatele. Pro zápis do souboru (souborů) využívám knihovnu au.com.bytecode.opencsv.CSVWriter.

XLS a XLSX

Stejně jako převod do CSV funguje i převodník do formátu XLS, potažmo XLSX. Využívá se zde stejné knihovny jako při vstupu těchto dvou formátů – Apache Poi. Průchod zploštělým objektem je zcela totožný jako v předchozím případě, data se zde jen neukládají do řetězce, ale do pracovního sešitu (WorkBook), listů (Sheet), řádků (Row) a buněk (Cell). Pracovní sešit je buď typu „XSSFWorkbook“ nebo „HSSFWorkbook“ podle typu výstupu. Jednotlivé „tabulky“ zploštělé proměnné zde mohou být reprezentovány v jednom sešitu pod sebou, nebo ve více sešitech, to určuje parametr konstruktoru „intoOneFile“ typu Boolean.

XML

Výstup souboru ve formátu XML využívám funkci třídy XML již zmíněné externí knihovny json.org. Funkce „XLM.toString()“ má za vstup formát JSON, který převede na výstupní řetězec ve formátu XML, tento výstup přesně podle vstupu. V případě neobaleného JSONu by tak mohl být výstupní soubor nekorektní. Pro při každém vytváření souboru přidám kořenový tag „<xmlDocument>“ na konci souboru „</xmlDocument>“. Tento kořenový tag zajistí, že budou všechny další tagy obaleny alespoň v tomto kořenovém. Výstup výše zmíněné funkce je sice XML řetězec, ale ten není správně odřádkován a odsazen, proto jsem pro lepší přehlednost souboru do funkce převodu zakomponoval i funkce, které tento řetězec dokáží správně formátovat. Výsledný formátovaný řetězec se poté zapíše do souboru zadaném uživatelem.

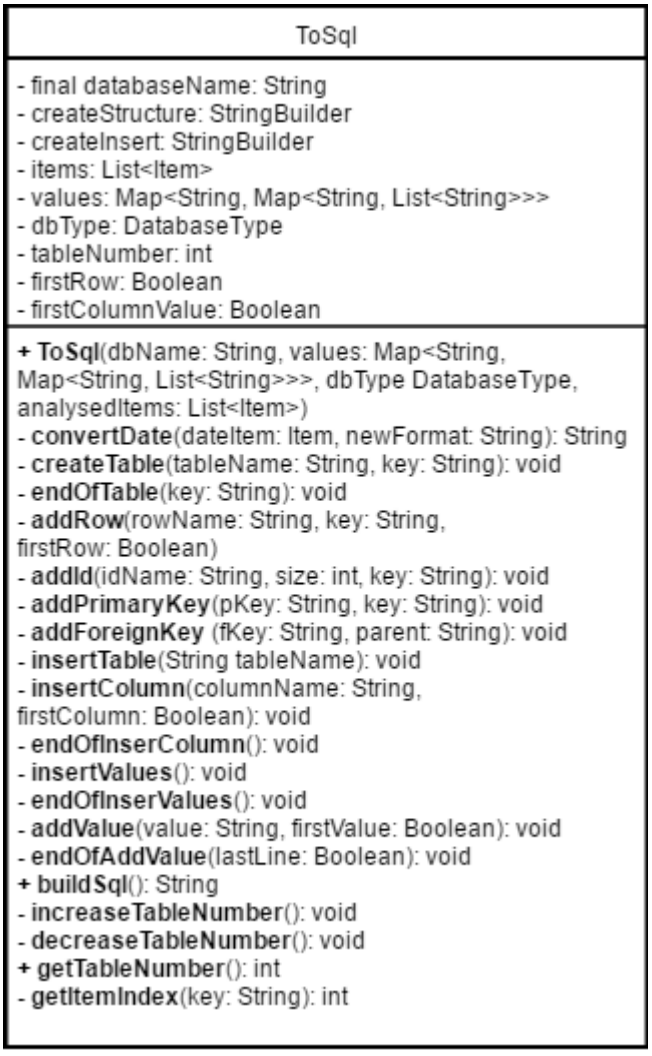
JSON

Vzhledem k tomu, že mezi-formát je typu JSON, tak již nemusím nic převádět. Pro uložení do výstupního souboru jsem si vytvořil třídu „SaveJson“. Veřejná metoda „saveJsonFile“ se vstupem v podobě mezi-formátu, výstupní cesty a názvu výstupního souboru, se stará o kompletní zápis do

souboru. Při zápisu se převádí obecný vstupní objekt na typ JSON objekt nebo pole, podle toho, jakého typu je. Poté už se využívá jen metody „toString“, která převede daný objekt na řetězec a umožní zadat, jak velké má být odsazení mezi jednotlivými vrstvami JSONu, díky tomu bude výstup správně formátován a dobře čitelný.

SQL

Pro vytvoření SQL výstupu (SQL insert souboru) jsem si vytvořil třídu „ToSql“ a několik dalších pomocných tříd, které popíši dále v textu. Pro převod na výstupní formát se využívá veřejná metoda „buildSql“, ta postupně volá další metody potřebné k převodu na SQL výstup. Pro spuštění převodu je nutné znát několik informací. Klasicky potřebuje znát typ databáze pro výstupní soubor a název databáze. Tyto dva údaje si zjistím od uživatele. Zbylé údaje byly získány při předchozí analýze a zploštění mezi-formátu JSON.



Obrázek 23 Diagram třídy ToSql

Ve výše uvedeném obrázku můžete vidět diagram třídy „ToSql“, její konstruktor si žádá pouze tři parametry – jméno databáze, zploštělé údaje a analyzované prvky s výsledky. Při vytváření databázového vstupu využívám třídní proměnné „createStructure“ a „createInsert“. Do těchto dvou proměnných postupně vkládám řetězce s názvy tabulek, jejich datové typy, velikostní limity atd. Jak již

prozrazují jejich názvy, jedna proměnná představuje strukturu SQL databáze a druhá pak SQL Insert do této vytvořené struktury.

Veřejná funkce „buildSql“ sestavuje obě proměnné najednou při průchodu zploštěné proměnné, přitom volá privátní funkce své třídy. Právě ty vkládají řetězce do zmiňovaných dvou třídních proměnných. Průchod zploštěné proměnné probíhá obdobně, jako tomu bylo v převodníku CSV nebo XLSX. Průchod probíhá v několika úrovních, na té první se vytváří databázové tabulka. V další se postupně vytvářejí všechny sloupce aktuální tabulky spolu s jejich datovými typy a limitací v případě řetězce. Přitom zde musím rozlišovat, zda se jedná o první, prostřední nebo konečný sloupec. V dalším průchodu se začnou vytvářet jednotlivé hodnoty, které se budou vkládat ke správným sloupcům.

O samotné vkládání hodnot do proměnných se stará řada mnou vytvořených metod, ty mají ve většině případu jako vstupní parametr aktuálně procházený prvek (jeho klíč), případně další potřebný parametr. Tyto privátní metody využívání několik mnou vytvořených tříd, konkrétně „SqlStatement“ a „DatabaseTypes“. Obě třídy se starají o mapování řetězců mezi různými databázemi tak, abych nemusel vytvářet vždy novou metodu při změně typu databáze. Třída „DatabaseTypes“ již byla popsána v předchozí kapitole 8.1.2 MySQL Dump, dále tedy budu popisovat pouze třídu „SqlStatement“.

Obrázek 24 Diagram třídy SqlStatement

Ve výše uvedeném obrázku můžete vidět mnou vytvořenou třídu typu Enum, která má v sobě uloženy všechny potřebné fráze k vytvoření výstupního SQL souboru. Jednotlivé položky tohoto výčtu reprezentují jednotlivé fráze. Například položka „CreateTable“ reprezentuje frázi "CREATE TABLE `%s` („ čili vytvoření databázové tabulky. Sada po sobě jdoucích znaků *%s*, reprezentuje proměnnou, která v tomto případě bude název tabulky.

Třídní proměnné pak reprezentují různé typy databází, ty jsou zde uvedeny z důvodu různého zápisu v různých typech databází. Například při zápisu proměnného řetězce se v MySql používá datový typ „Varchar“, kdežto v Oracle databázi se používá „Varchar2“. Já jsem tento problém řešil různým řetězcem pro každý typ databáze. Z obrázku tedy lze vyčíst, že již zmíněná položka „CreateTable“ může nabývat 4 různých hodnot, pro každou z databází. Určitý výstupní řetězec je přístupný přes funkci „getStatement“, ta má jako vstup proměnnou typu DatabaseType (databázový typ). Podle této proměnné se určí příslušní řetězec, který se má vrátit touto funkcí.

Nejkomplexnější metoda generující SQL výstup je funkce „addRow“ třídy „toSql“. Ta má tři vstupní parametry – jméno příslušného řádku, klíč a parametr typu Boolean, který určuje, zda je řádek první nebo ne. Tato metoda je důležitá především proto, že spojuje výsledky analýzy spolu s daty zploštělého formátu.

Po vytvoření obou proměnných „createStructure“ a „createInser“ je funkce „buildSql“ spojí a vrátí jako řetězec řídící třídě. Ta tento řetězec předá statické funkci „writeStringToFile“, třídy „FileWrite“, která jej zapíše do výstupního souboru s uživatelem zadaným názvem.

Všechny diagramy, které jsou součástí tohoto dokumentu (včetně kompletního třídního diagramu) naleznete v přílohách této bakalářské práce, ve složce „**Přílohy/Diagramy**“.

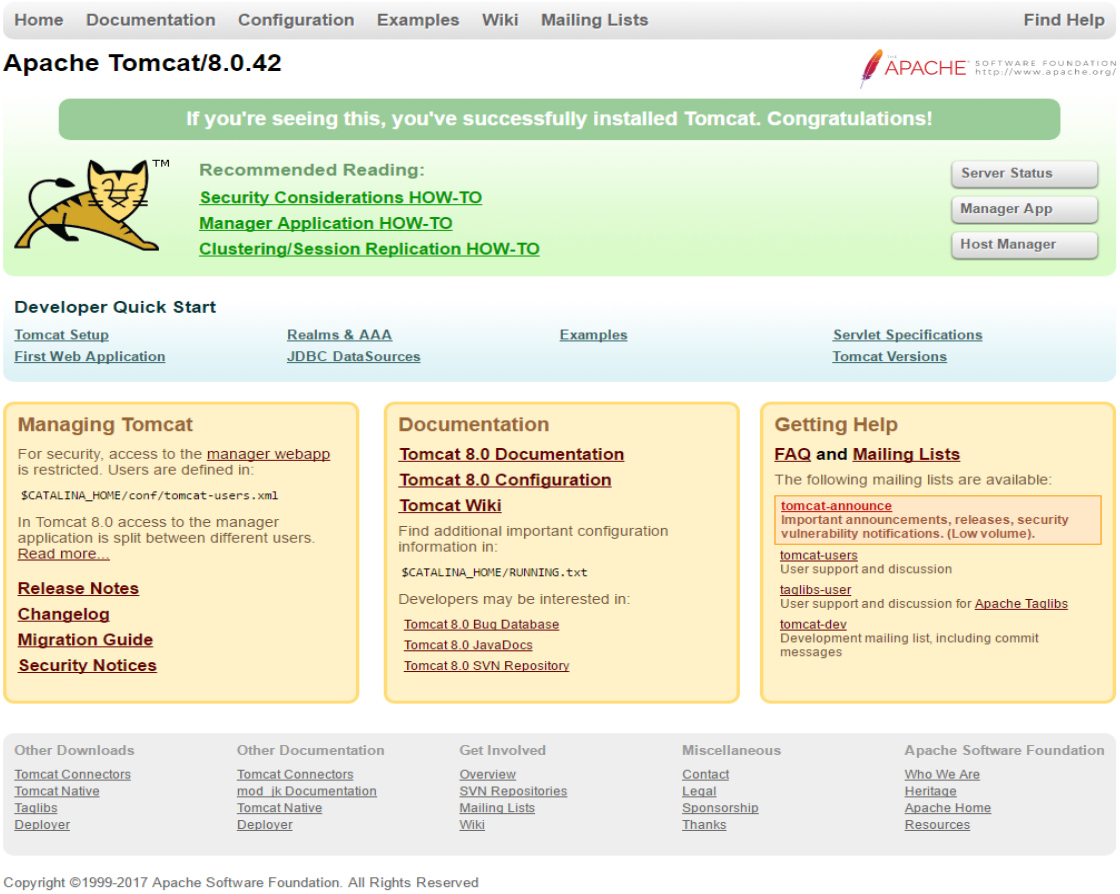
6 Nasazení aplikace na web

Pro nasazení aplikace na webové rozhraní jsem se rozhodl využít možnosti programovacího jazyka JAVA, konkrétně jsem využil frameworku Spring MVC ve spojení s webovým serverem Apache Tomcat (verze 8.0.42), vše je pak umístěno na servery Amazonu pomocí jejich webových služeb AWS.

6.1 Apache Tomcat

Apache Tomcat je open source webový server založený na programovacím jazyce JAVA, podporuje tak implementace Java Servlet, JavaServer Pages (JSP), Java Expression Language a Java WebSocket technologie. Tento webový server je vyvíjen již několik let a vyšel tak v mnoha verzích, nejnovější verzí je momentálně Tomcat 9, já ve své aplikaci využívám starší verzi Tomcat 8, z důvodu omezení mého vývojového prostředí, které novou verzi serveru zatím nepodporuje. [33]

Instalace serveru není nijak složitá, z webových stránek http://tomcat.apache.org/ stačí stáhnout příslušnou verzi aplikace a tu pak rozbalit na disku. Spuštění lze provést přímo v prostředí editoru NetBeans nebo příkazovým řádkem. Instalaci serveru lze provést přímo při instalaci prostředí NetBeans, kde si můžeme vybrat z více možností webových serverů. Při instalaci na platformu Linux lze využít příkazové řádky pro instalaci, konfiguraci i spuštění serveru. Po instalaci a puštění serveru se po zadání příslušné adresy do prohlížeče dostanete na domovskou stránku serveru (obrázek níže).



Obrázek 25 Výchozí stránka webového serveru Apache Tomcat 8.0.42

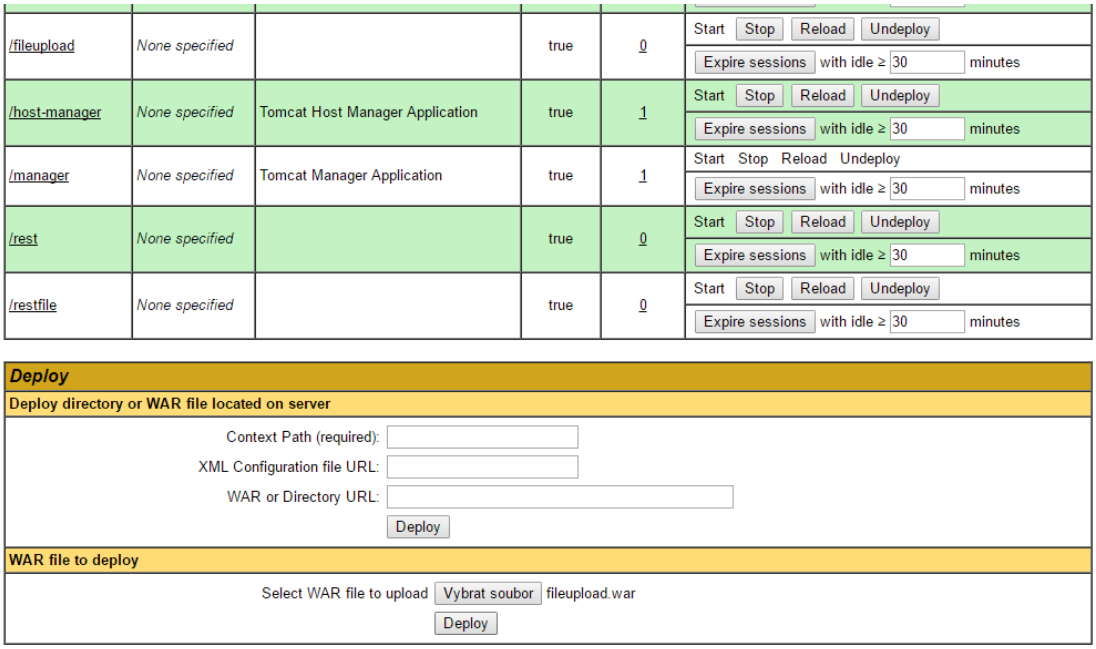
Výchozí port, na kterém server běží, je 8080, lze jej změnit v souboru „„„$CATALINA_HOME/conf/server.xml“ Jak je vidět na obrázku, domácí stránka nabízí spoustu možností, kam můžeme přejít, je zde umístěn odkaz na dokumentaci Apache Tomcat (provozované verze). Různé příklady použití (již vytvořených aplikací pro tento server) a spoustu dalšího. Nejdůležitější části jsou umístěny v pravé části, a to jsou tlačítka „Server Status“, „Manager App“ a „Host Manager“. Přístup do těchto sekcí je omezen přihlášením. V defaultním nastavení serveru se však nenachází žádní uživatele, proto je musíme přidat do souboru „$CATALINA_HOME/conf/tomcat-users.xml“. $CATALINA_HOME reprezentuje kořenovou složku, kde je umístěn server ApacheTomcat.

Do výše uvedeného souboru musíme přidat uživatele, jeho heslo a role, ty reprezentují funkce, které tento uživatel může provádět. Níže uvedený příklad přidá uživatele „tomcat“ s heslem „tomcat“, který bude moct přistupovat do všech tří nastavení a bude je moct měnit podle svého uvážení.

```
<role rolename="admin"/>
    <role rolename="admin-gui"/>
    <role rolename="manager-gui"/>
    <user password="tomcat" roles="manager-script,admin,manager-gui,admin-gui" username="tomcat"/>
</tomcat-users>
```

Po kliknutí na první tlačítko se dostaneme na stránku zobrazující aktuální informace o chodu serveru. Jsou zde uvedeny informace o systému, na kterém je server provozován, seznam běžících aplikací, využité prostředky systému a mnoho dalšího. Po kliknutí na odkaz „List Applications“ se dostaneme na stránku seznamu všech nasazených aplikací, můžeme je zde i editovat – spustit, zastavit atd.

Při kliknutí na druhé tlačítko „Manager App“ se dostaneme na totožnou stránku s výpisem nasazených aplikací. Poslední tlačítko „Host Manager“ Vás přesměruje na stránku virtuálního host manageru, která umožní nasazení více webových stránek (DNS) na jedinou instanci serveru.



Obrázek 26 Nasazení aplikace v Apache Tomcat

Pro nasazení vytvořené webové aplikace je třeba vložit kompilovanou aplikaci do serveru. Abychom to mohli udělat, musíme přejít do „správce aplikací“. Za výpisem všech nasazených aplikací nalezneme oddíl pro nasazení WAR souboru „WAR file to deploy“, ten jsme získali při kompilaci programu. Po nahrání příslušného souboru je aplikace nasazená na URL specifikovanou při jejím vytváření.

Umístění souborů Apache Tomcat serveru na Linuxu:

CATALINA_BASE="/usr/share/tomcat"
CATALINA_HOME="/usr/share/tomcat"
JASPER_HOME="/usr/share/tomcat"
CATALINA_TMPDIR="/var/cache/tomcat/temp"

Název složky „tomcat“ je vždy doplněn o instalovanou verzi tohoto serveru. Například „tomcat8“

6.2 Spring MVC

Pro vytvoření samotné webové aplikace jsem využil již zmíněný framework Spring MVC. Tento framework vychází z návrhového modelu MVC, rozděluje tedy uživatelské rozhraní do tří nezávislých komponent (Model, View, Controller). Framework je postaven na „DispatcherServlet“, který rozesílá jednotlivé požadavky uživatele z webu na určité „handlery“ vytvořené programátorem. [34] Při práci s tímto frameworkem se využívají nejrůznější anotace. Mezi základní patří například „@Controller“, „@RequestMapping“, „@ModelAttribute“ a další.

První anotace značí, že je daná třída kontrolorem, který řídí běh webové stránky. Pro mapování webové adresy se využívá druhá zmíněná anotace, díky ní můžeme upravovat URL adresu podle toho, jak potřebujeme. Poslední anotace označuje ty atributy, které chceme poslat z View do Controlleru nebo naopak tak, abychom například mohli vypisovat chybové hlášky programu, nebo informace o průběhu programu.

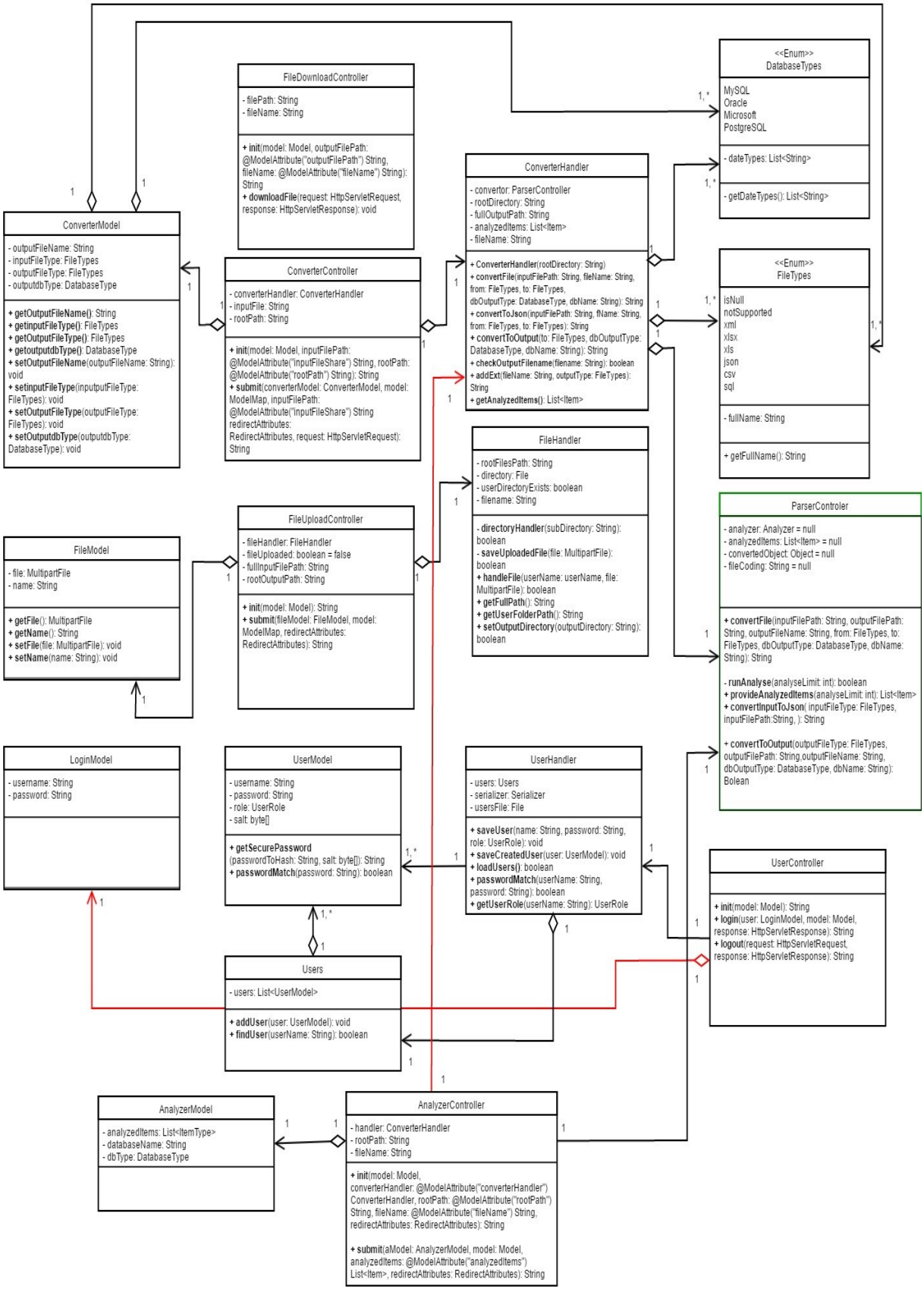
Při práci s tímto frameworkem jsem musel nejprve upravit již zmíněný konfigurační soubor „DispatcherServlet“. Do tohoto souboru jsem přidal následující řádek:

```
<context:component-scan base-package="controllers" />
```

Tento řádek zajistil to, že jsou brány v potaz anotace definující URL adresu **@RequestMapping**, a to v jednotlivých mnou vytvořených kontrolérech, umístěných v balíčku s názvem „**controllers“**.

Pro správné fungování webové aplikace bylo nezbytné přidání aplikace Parseru, který byl popsán v kapitole 5. Ten bylo třeba zkompilovat do JAR souboru spolu se všemi na importovanými knihovnami. To lze provést buď pomocí konzole a příslušných příkladů, nebo přímo v prostředí NetBeans, po patřičné úpravě konfiguračního souboru kompilátoru „build.xml“ (Upravený soubor je součástí přílohy této bakalářské práce).

Po připojení aplikace parseru jako externí knihovnu webové aplikace, lze využívat všechny veřejné funkce. V mé webové aplikaci se viditelně využívají jen tři třídy aplikace parseru – „ParserController“, „FileTypes“, „DatabaseTypes“. První z nich umožňuje spuštění převodu vstupního souboru na výstupní při zadání požadovaných parametrů. Tato třída také při úspěšném převodu vytvoří výstupní soubor na zadanou výstupní cestu. Zbylé dvě třídy jsou typu ENUM a slouží k zobrazení podporovaných datových typů uživateli a také ke kontrole při vstupu uživatele.



Obrázek 27 Třídní diagram webové aplikace

Výše uvedený obrázek 27 reprezentuje třídní diagram webové aplikace. Na tomto obrázku můžeme vidět propojení aplikací parseru a webové aplikace, konkrétně třídou „ConverterHandler“, která se stará o řízení převodu s uživatelem zadanými parametry. V třídě jsou také naimplementovány metody pro kontrolu výstupního souboru, konkrétně kontroluje, zda má zadaný název příslušnou příponu či nikoli. V případě, že přípona chybí, metoda ji přidá podle výstupního typu, který je pro uživatele povinným údajem při spuštění převodníku.

Instance této třídy se využívá v třídě „ConverterController“, která se stará o propojení s prezentační vrstvou, která zobrazuje a získává informace od uživatele. Třída přitom kontroluje veškerý vstup uživatele a vyžaduje opětovné zadání nekorektních, nebo nulových hodnot, kterou jsou povinné. Pro správný chod aplikace je nejprve potřeba získat vstupní soubor uživatele. K tomu slouží třída „FileUploadController“ a „FileHandler“.

Třída „FileUploadController“ se opět stará o správné zobrazení a získání dat od uživatele. Přitom provádí kontroly vstupu a případně vypisuje chybové hlášky nebo jiné důležité informace. Třída „FileHandler“ se pak stará o samotné uložení souboru do správného umístění, podle toho, co jí předchozí třída předala.

Pro samotné uložení souboru na server se využívá třída „MultipartFile“ zabudovaná ve Spring frameworku a interní funkce „FileCopyUtils.copy“. Nahrání souboru uživatelem probíhá v podstatě tak, že uživatel vybere soubor ze svého počítače a po stisknutí tlačítka „Upload“ se soubor nahraje právě do instance třídy „MultipartFile“, která je zatím v operační paměti. Z operační paměti se soubor dostane funkcí „FileCopyUtils.copy“, ta překopíruje soubor z paměti na zadané místo na disku. Třídě „ConverterController“ se pak předá cesta k souboru a jeho název.

Po nahrání souboru se spustí již popsaná část převodu souboru. Po úspěšném převedení na výstup předá třída „ConverterController“ cestu k výstupnímu souboru třídě starající se o zobrazení stránky pro stáhnutí souboru zpět do uživatelova zařízení.

Tato třída nemá za úkol nic jiného než vzít předané parametry předchozího Controlleru a v případě stisknutí tlačítka pro stáhnutí souboru zjistit, zda existuje výstupní soubor. Poté tato třída daný soubor načte do paměti a zkopíruje jej do podoby HttpServletResponse, která bude zobrazena uživateli jako vyskakovací okno s nastavením pro uložení souboru do jeho zařízení.

Komunikace mezi jednotlivými Controllery je zajištěna předáváním atributů modelu, konkrétně pomocí anotace „@ModelAttribute“. Přechod mezi jednotlivými kroky webové aplikace se provádí pomocí přesměrováním stránek (redirect). Veškerá data (proměnné) jsou uložena v instancích tříd modelů.

Pro vykreslení HTML stránek se zde využívají JSP sobory, do kterých lze přímo vkládat HTML tagy, ale také kusy kódu, které napomohou při vykreslení dat uživateli. Příkladem takového kódu je konstrukce foreach, která mi pomáhá vypsat možnosti pro výběr datových typů.

6.2.1 Uživatelské účty a uživatelé

Jedna z dalších mnou naimplementovaných části aplikace je řízení uživatelských účtů. Pro správu účtů využívám XML databázi, kterou si vytvářím pomocí dvou tříd své aplikace. Ve třídách „UserModel“ a „Users“ využívám frameworku **Simple XML Serialization**, kterým tyto třídy zapisuji (načítám) do souboru. Třída „UserModel“ má jednoduchou strukturu, je zde pouze jméno uživatele, heslo a jeho role

(ta je definovaná v ENUM „UserRole“). Heslo je bezprostředně po zadání, a potvrzení registračního formuláře, převedeno do podoby hashe. Převod provádím pomocí základních bezpečnostních funkcí v Javě. Pro obtížnější rozluštění hesla je ke vstupnímu řetězci připojen i tzv. salt.

Při pokusu o přihlášení se tak načítá soubor z disku, ve kterém jsou uloženy výše zmíněné údaje. Porovnává se zde jméno uživatele a hash jeho hesla s hashem, vygenerovaném při přihlašování. Role uživatele se využívá se využívá pro zobrazení různých funkcí online aplikace. Níže je uveden soubor s uživatelskými účty.

```
<users>
   <userList class="java.util.ArrayList">
      <user>
         <name>user1</name>

<password>367ff3403ea4348780eb76fe82efe3d70fc2b3d55ec2c01e8a293e0fd0d11f5c<
/password>
         <role>normal</role>
         <salt length="16">-44, 103, 118, 126, 45, -35, 69, -68, -60, -37,
51, 64, 30, 50, -103, -88</salt>
      </user>
      <user>
         <name>user2</name>

<password>db16d337826b5ee84a82eaf2dc895dc911132ce71425266d5f9e902fdc7c862d<
/password>
         <role>admin</role>
         <salt length="16">40, -101, -106, 98, 6, -66, -119, -21, 120, 41, -
108, 127, 66, -53, 86, -44</salt>
      </user>
      <user>
         <name>user3</name>

<password>e102ccb9a01f8007ec0e81cbae2df59414413ea66d42044ebbae30bfe8777078<
/password>
         <role>normal</role>
         <salt length="16">-89, 111, 85, 21, -67, 43, 58, 49, -105, 92, 14,
-49, 5, 17, 64, -18</salt>
      </user>
      <user>
         <name>kol0281</name>

<password>f00ab987e9bad31b8ae32719682519df75c41f8c0726e3d0af97ecf7b35dbe78<
/password>
         <role>admin</role>
         <salt length="16">-68, -26, 45, 124, 102, -114, -39, 125, -113, -
62, -87, 59, 48, 73, -43, -110</salt>
      </user>
   </userList>
</users>
```

6.3 Nasazení na server Amazonu

Pro nasazení aplikace jsem zvolil již zmíněné online služby Amazonu AWS – Amazon Web Services. Zvolil jsem je kvůli možnosti vytvoření bezplatného účtu na období 12 měsíců. Tyto online služby umožnují vytvoření vlastního veřejného serveru, s instalací zvoleného operačního systému (Windows nebo různé instance Linuxu). Lze spustit i databázový server, já však této možnosti nevyužívám. Při samotné instalaci serveru si můžeme vybrat nejrůznější nastavení toho, co chceme na konkrétním server nainstalovat. Já jsem například vybral instalaci Javy, dá se také vybrat instalace Webového serveru z nabídky.

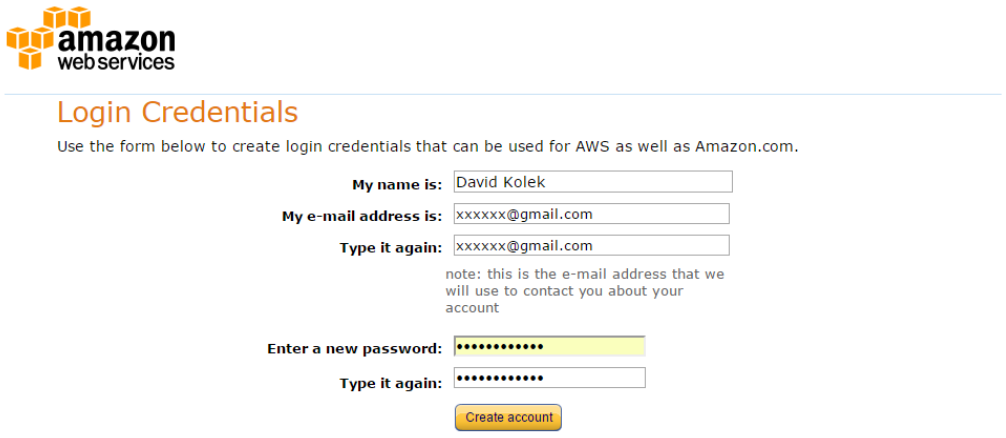
Bezplatná licence má však svá omezení. Tím největším je omezení výpočetního výkonu instance serveru. V bezplatném účtu totiž lze využít pouze instance t2.nano nebo t2.micro. T2.micro nabízí lepší výpočetní výkon než verze nano, a to v podobě jednoho procesoru a 1 GB paměti RAM. Omezení operační paměti serveru má za následek nemožnost zpracování velkých datových souborů mou aplikací. To totiž k rozbalení například 50 MB souboru potřebuje více operační paměti než 1 GB. Menší datové soubory však online rozhraní aplikace zvládne bez problému. V placeném účtu se účtuje cena za hodinu procesorového času, kde každá různá instance serveru stojí jinou cenu. Více o různých instancích serverů a jejich cenách naleznete na stránkách AWS. [35]

6.3.1 Vytvoření účtu na AWS a založení instance serveru

Prvním krokem k nasazení je vytvoření účtu ve službě AWS na webové stránce www.aws.amazon.com. Zde stiskneme v pravé horní části tlačítko „Sing In to the Console“.

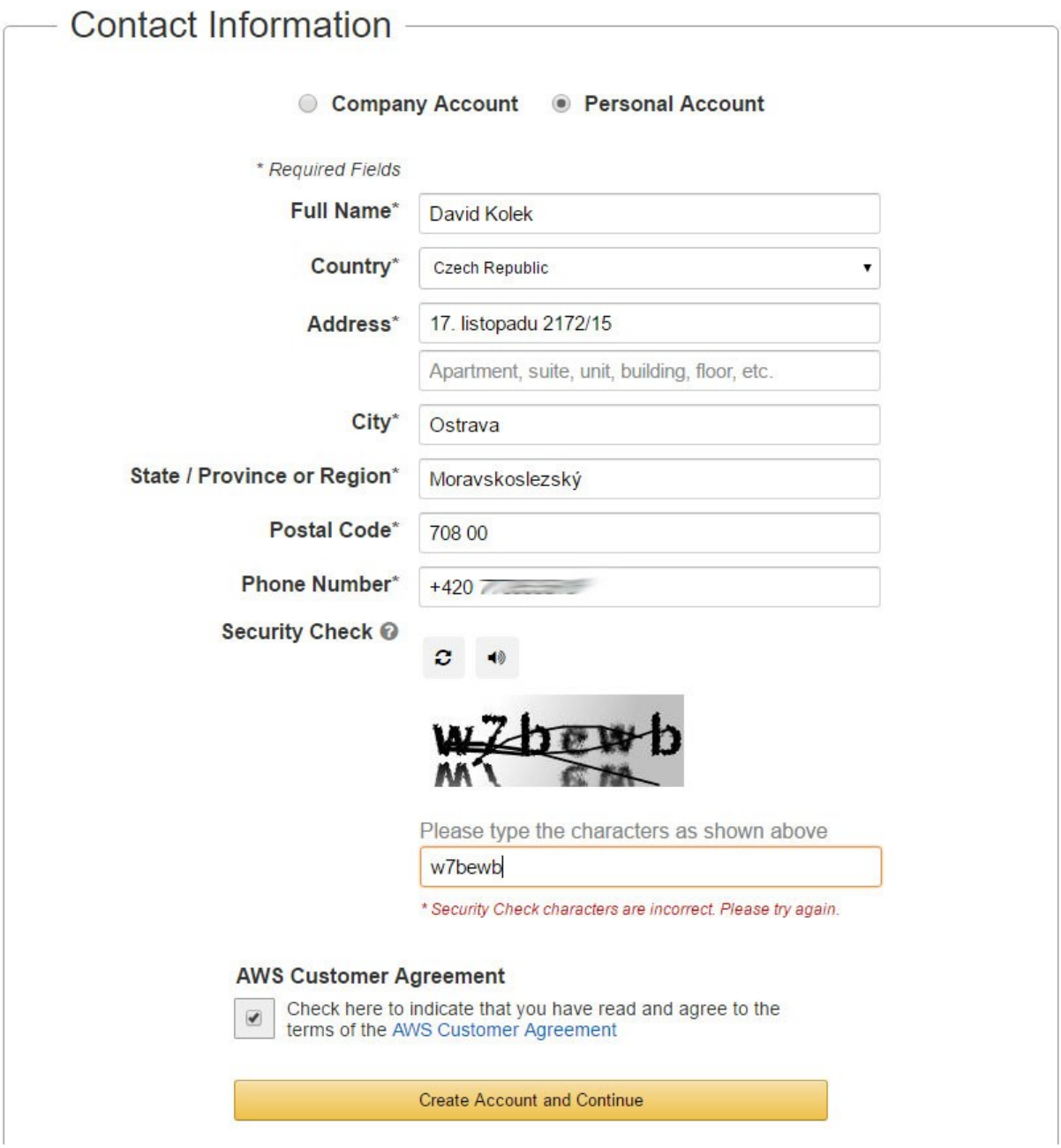
Obrázek 28 Vytvoření účtu na AWS

Po přesměrování se dostaneme na výše uvedenou stránku, kde vyplníme svou emailovou adresu, zaškrtneme políčko „I am a new user“ a stiskneme tlačítko „Sing in using secure server“.



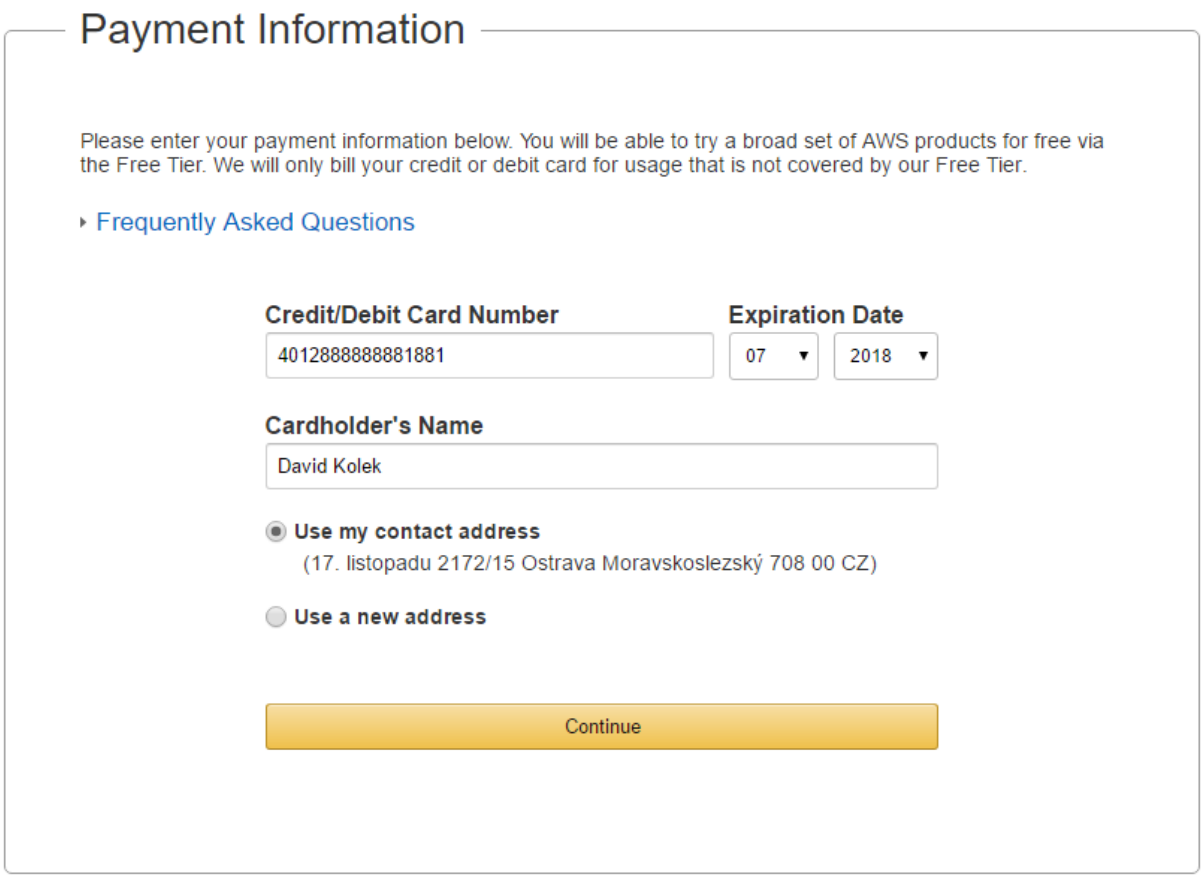
Obrázek 29 Registrace na AWS základní údaje

Poté se již dostaneme na stránku, kde je potřeba vyplnit několik základních údajů, viz obrázek 29. Po vyplnění těchto údajů stiskneme tlačítko „Create account“.



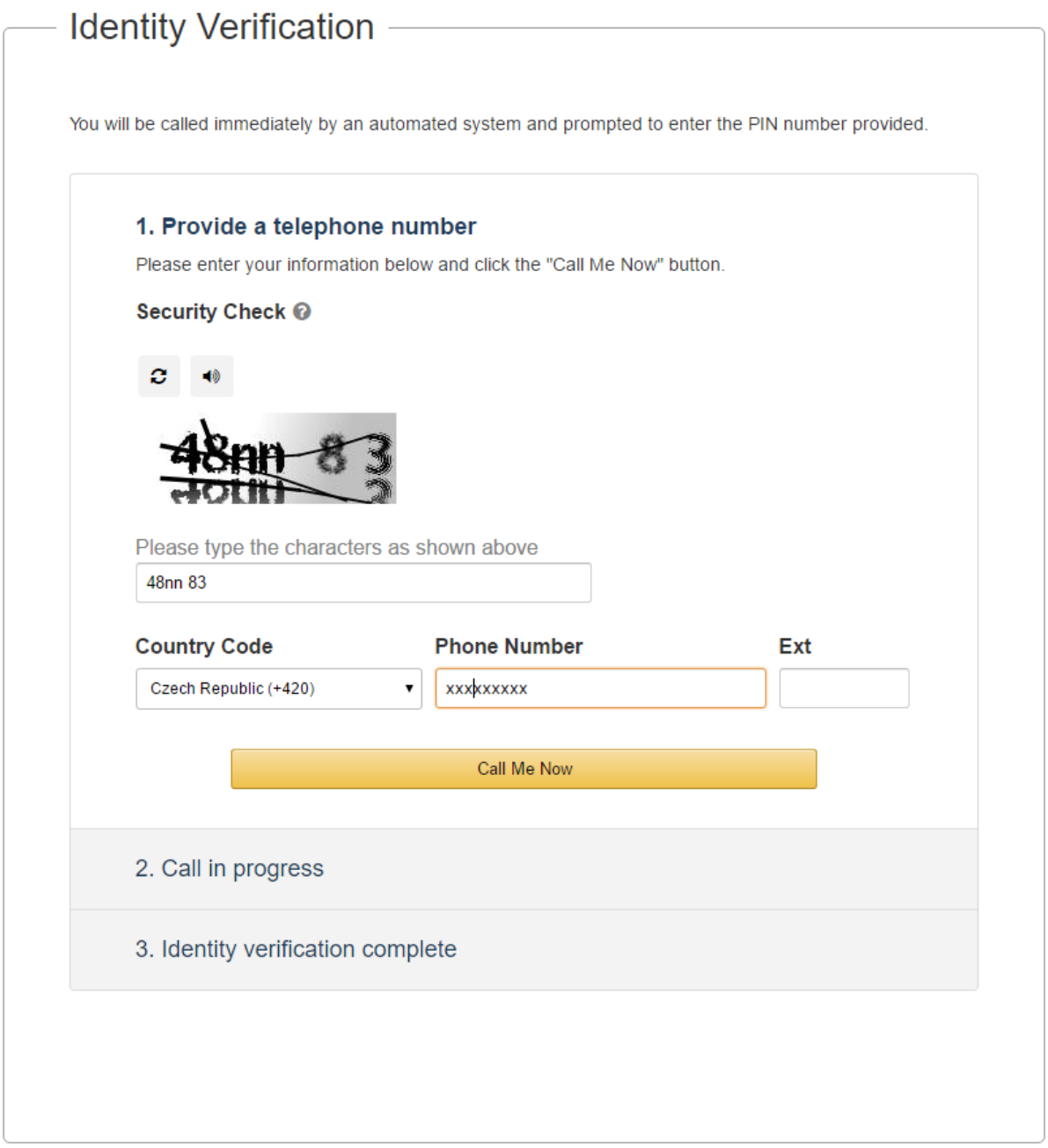
Obrázek 30 Registrace na AWS Osobní údaje

V dalším kroku se dostaneme na stránku, kde je potřeba vyplnit osobní údaje, všechny vyplníme, zaškrtneme box (souhlasíme s podmínkami AWS) a pokračujeme dále stisknutím tlačítka „Create Account and Continue“. V horní části je možnost výběru registrace buďto firemního, nebo osobního účtu, já jsem vybral osobní účet. V možnosti firemního účtu je potřeba vyplnit více údajů.



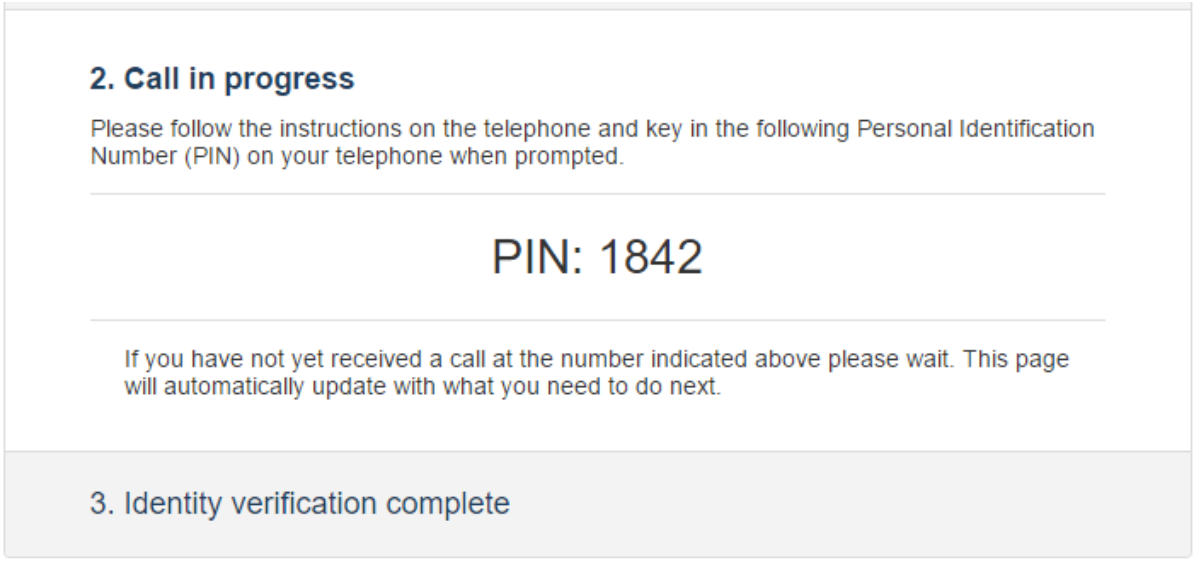
Obrázek 31 Registrace na AWS platební údaje

Dalším krokem při tvorbě nového účtu, je zadání platebních údajů. Na výše uvedeném obrázku vidíme, co vše je nutné vyplnit. Musíme zde vložit číslo a datum vypršení platné platební karty a jméno jejího majitele. I když se zde zadává platná karta, nemusíme se bát, že by se při registraci účtu stáhla z karty jakákoli částka. Platby se provádějí na základě spuštěných služeb, které se vybírají až po registraci uživatelem. Po vložení všech údajů stiskneme tlačítko „Continue“.



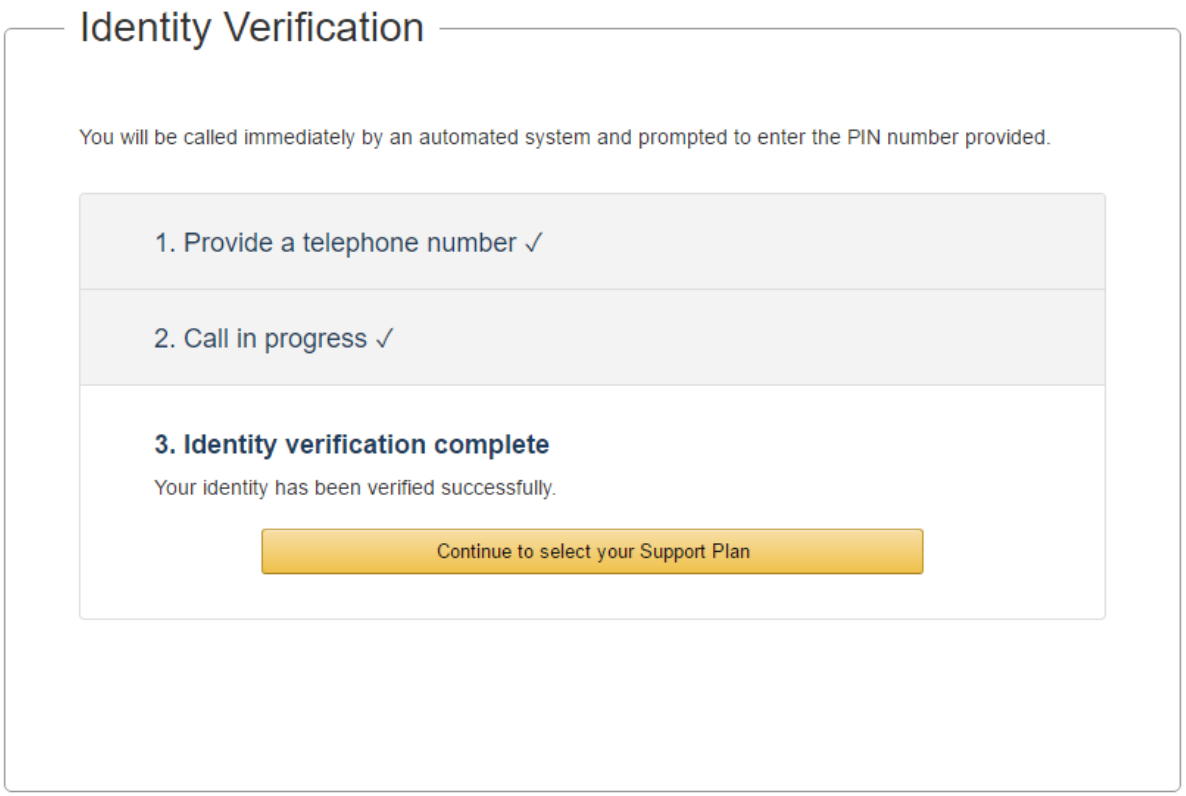
Obrázek 32 Registrace AWS verifikace údajů

Na obrázku 32 se dostáváme k dalšímu kroku, a to je verifikace údajů. Zde si zkontrolujeme zadané telefonní číslo a vyplníme bezpečnostní kód. Poté následuje verifikace telefonního čísla, která se provádí po stisknutí tlačítka „Call Me Now“.



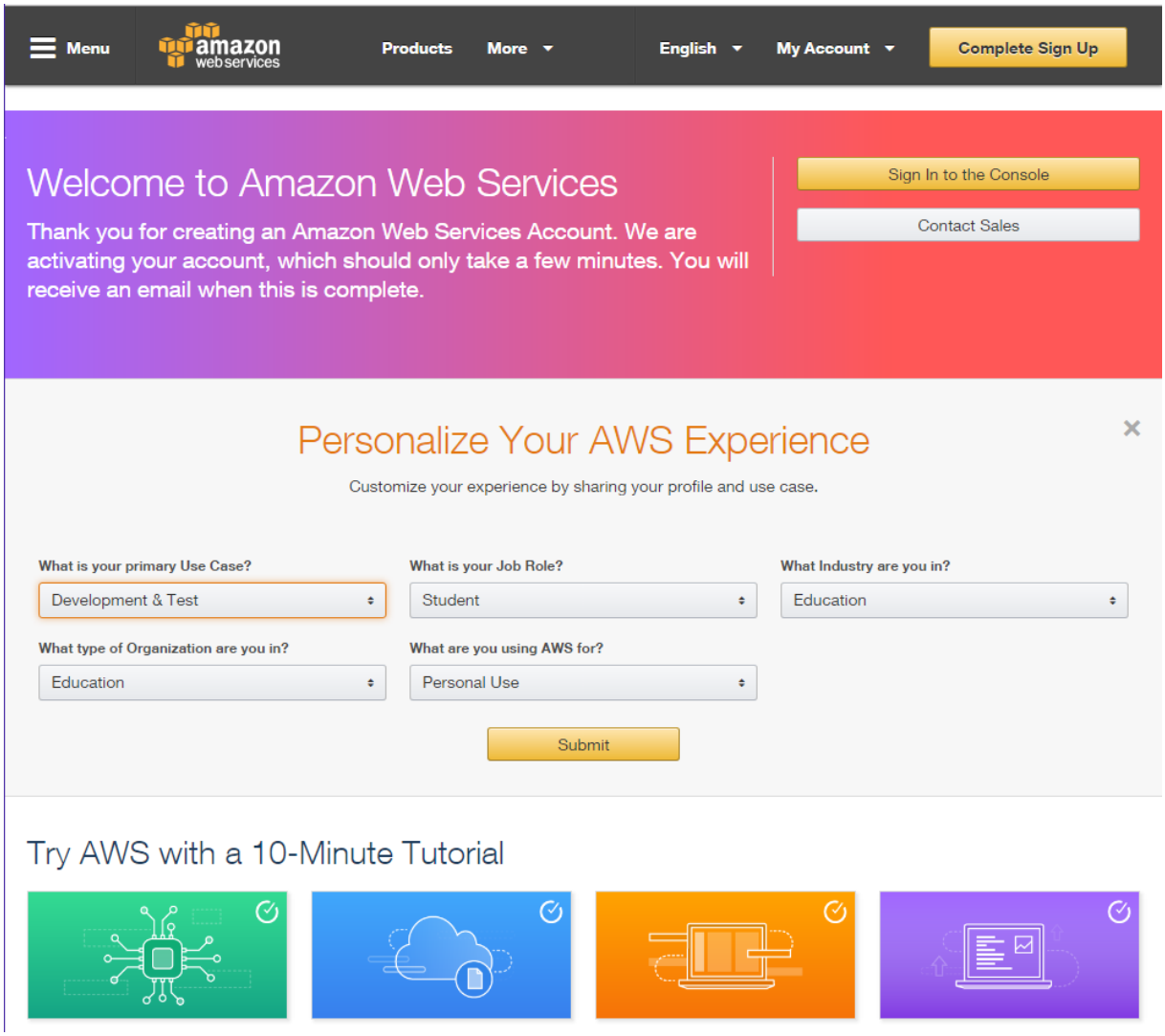
Obrázek 33 Registrace AWS verifikace hovorem

Po stisknutí tlačítka se nám rozbalí oddíl „Call in progres“ a zároveň nám začne volat automat z Amazonu, na dříve uvedené telefonní číslo. V telefonu nám sdělí, že máme pomocí zadávacího panelu v našem telefonu zadat PIN kód, který se nám zobrazil na obrazovce (viz obrázek 33). Po úspěšném zadání kódu do telefonu se nám automaticky sbalí panel „Call in progress“ a zároveň se odbalí panel „Identity cerification complete“. Zde se nám zobrazí informace o úspěšné verifikaci telefonu a také tlačítko pro pokračování dále (viz obrázek níže).



Obrázek 34 Registrace AWS úspěšné ověření telefonu

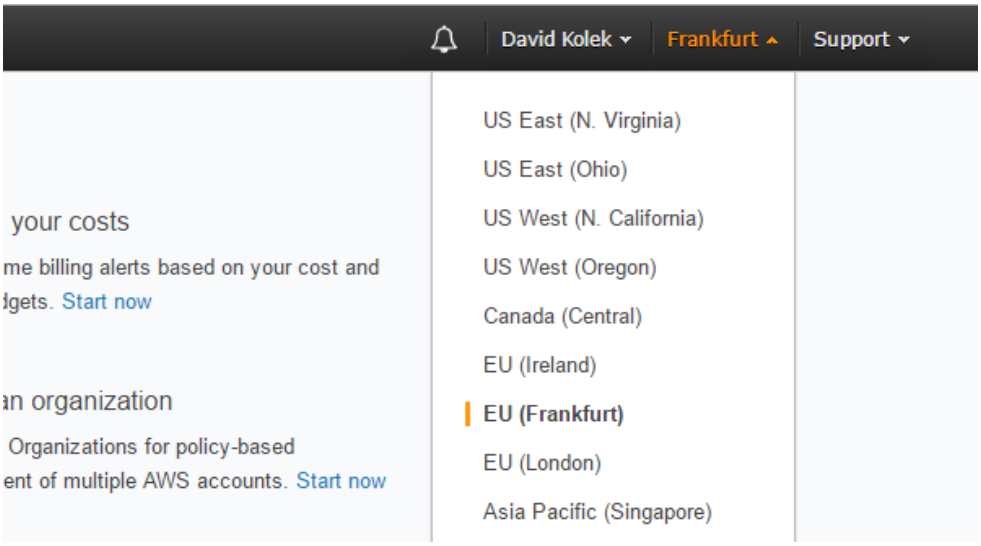
V dalším kroku si vybereme podporu pro svůj účet, já vybírám typ Basic, protože lepší nepotřebuji. Můžeme si však vybrat hned z několika typů podpory, u každé z nich je napsáno, co zahrnuje a za jakou cenu je nabízena. Podpora typu Basic je bezplatná a zahrnuje přístup do komunitního fóra AWS, podporu pro nastavení účtu a podporu platebních metod. Po vybrání příslušné podpory stiskneme tlačítko „continue“.



Obrázek 35 AWS personalizace profilu

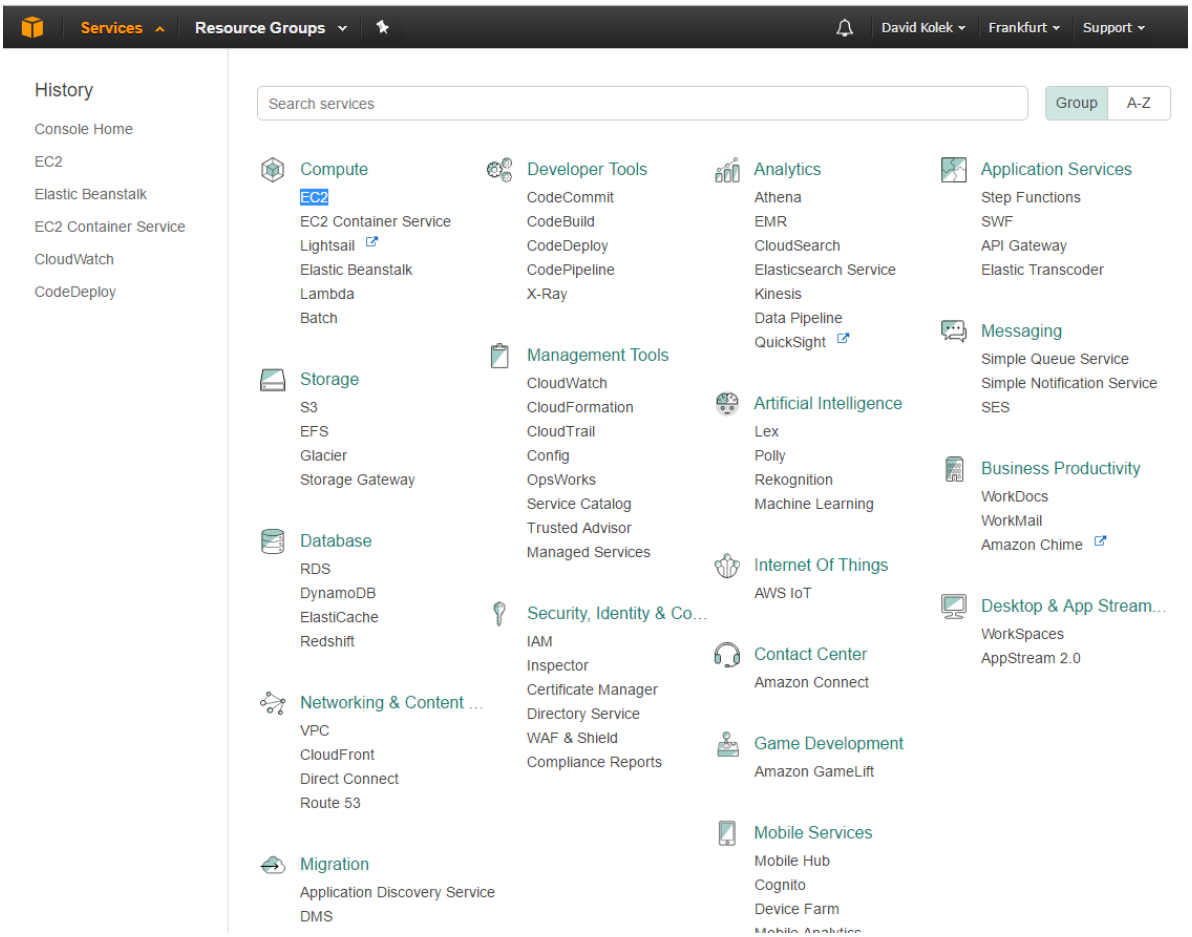
Poté se již dostaneme zpět na stránky AWS, kde se nás systém zeptá, k čemu jej chceme využívat. Zadáme informace a stiskneme tlačítko „Submit“. Tímto krokem končí registrace a můžeme se přihlásit do administrativní konzole. Stisknutím tlačítka „Complete Sing UP“ v horní části stránky. Poté zadáme email a heslo a přihlásíme se.

6.3.2 Vytvoření prostředí pro aplikaci



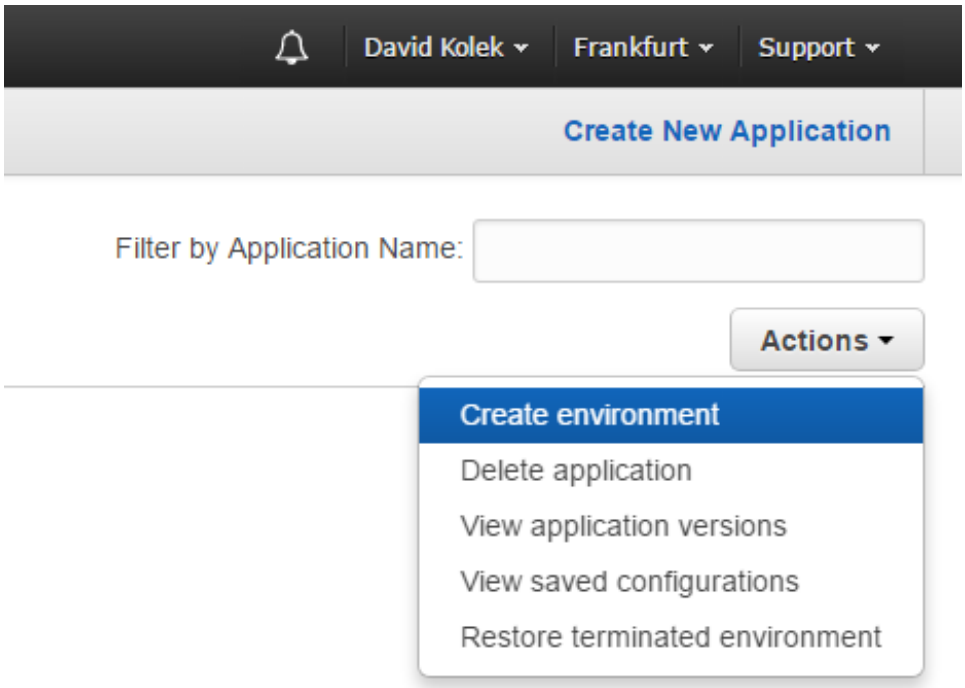
Obrázek 36 ASW výběr lokality serveru

Po úspěšném přihlášení přichází na řadu vytvoření instance serveru. V prvé řadě však musíme vybrat lokalitu, ve které má být tento server vytvořen. To se provádí na úvodní straně, kde si v pravé horní části rozbalíme menu s lokalitami a vybereme požadovanou. Já jsem vybral nejbližší České republice „EU (Frankfurt)“. Po výběru lokality přejdeme k vytvoření serveru, to se provádí v záložce „Services“ v levé horní straně stránky.



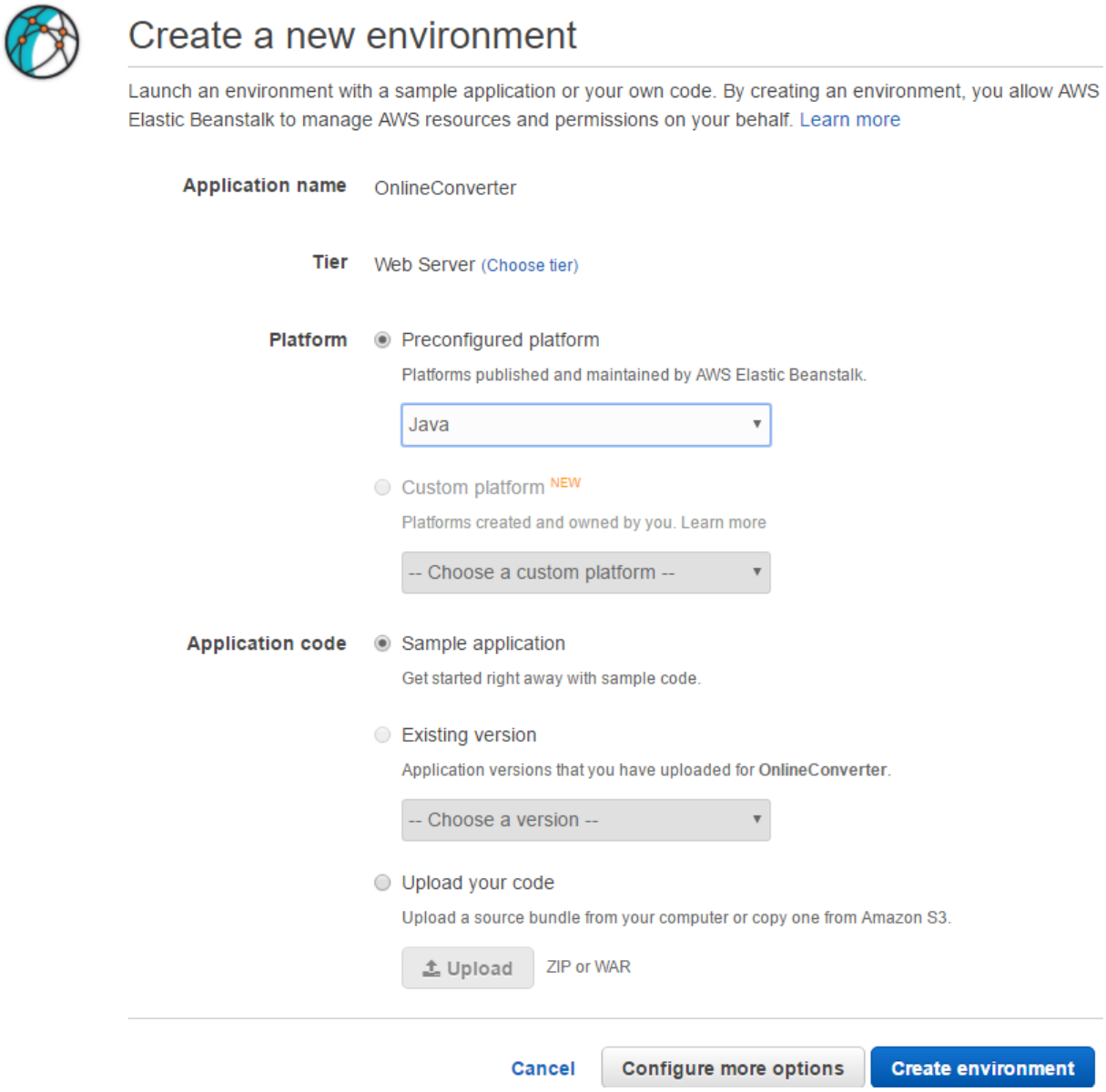
Obrázek 37 Výběrové menu záložky Services

Po stisknutí této záložky, nám vyjede rozsáhlé menu možností (viz obrázek 37). V tomto menu vybereme položku **Elastic Beanstalk**. Jak můžete vidět na obrázku 37, AWS nabízí nepřeberné množství služeb, které můžeme využít, některé jsou bezplatné, jiné jsou placené. K vybraným položkám tohoto menu se budu vracet v dalších krocích.



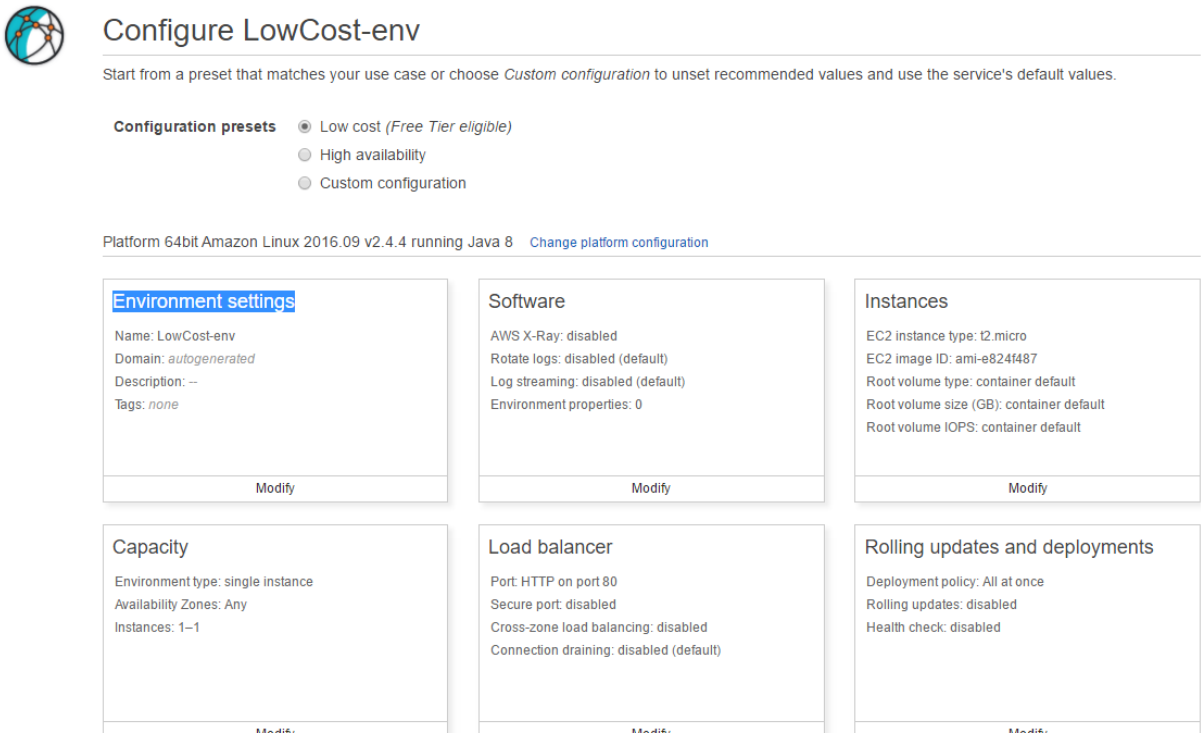
Obrázek 38 AWS vytvoření vlastního prostředí pro aplikaci

Po kliknutí na odkaz **Elastic Beanstalk** se dostaneme do menu této služby, pro vytvoření nové aplikace klikneme na výběrové menu „Actions“ (napravo v horní části) a vybereme položku „Create Enviroment“. Po stisknutí této položky nám vyskočí okno s možností výběru mezi „Web server environment“ a „Worker environment“. Vybereme první možnost a klikneme na tlačítko „Select“.



Obrázek 39 AWS vytvoření nového prostředí

Poté se dostaneme k nastavení vlastního prostředí pro aplikaci. Vybereme potřebné nastavení (Platforma JAVA) a klikneme na tlačítko „Configure more options“. Po kliknutí na toto tlačítko se dostaneme do podrobnějšího nastavení.

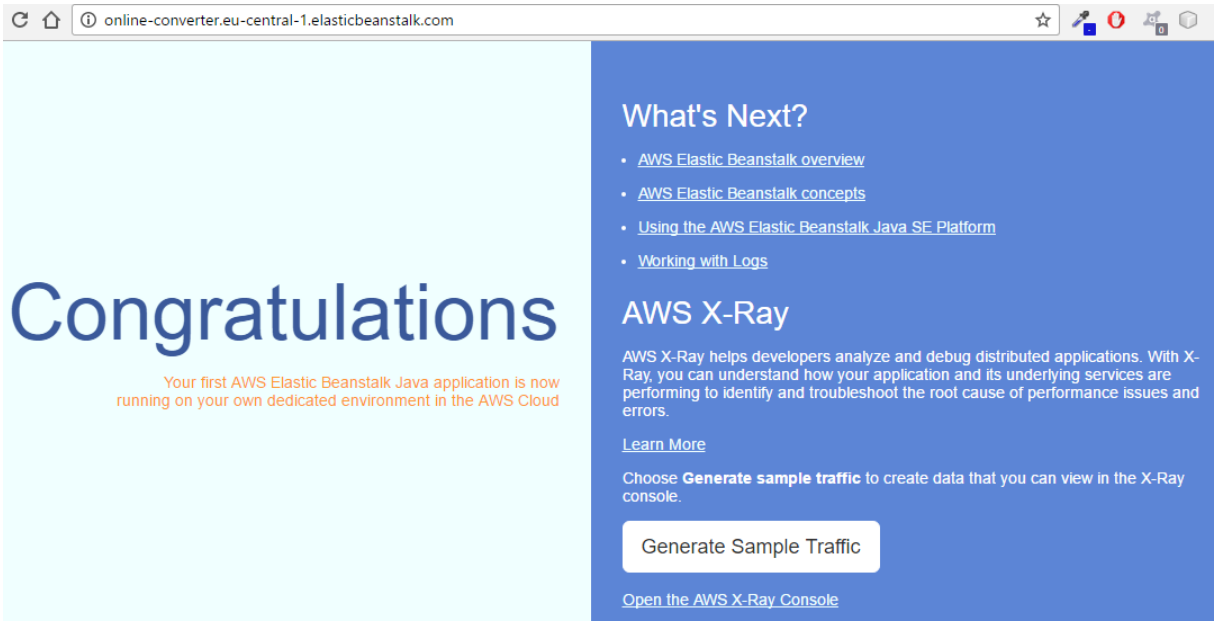


Obrázek 40 AWS podrobné nastavení nového prostředí

Na této stránce (obrázek 40) vidíme spoustu panelů s informacemi o vytvářeném prostředí. V panelu „Instances“ například vidíme, jakého typu je instance serveru (t2.micro). Jednotlivé záložky můžeme rozkliknout a nastavení zde změnit. Můžeme vylepšit výkon serveru výběrem silnější instance, ta však už není bezplatná. Nás však bude zajímat záložka „Enviroment settings“, v ní klikneme na tlačítko „Modify“.

Po stisknutí této záložky nám vyjede okno, ve kterém si specifikujeme jméno naší aplikace. V mém případě vyplním název „OnlineConverter“. Vyplníme i políčko „Domain“ - doména, na které aplikace poběží, já vyplňuji „online-converter“. Má aplikace tedy poběží na adrese „online-converter.eu-central-1.elasticbeanstalk.com“. Po potvrzení tlačítkem „save“ se dostaneme zpět na výběr nastavení, zde si můžeme změnit platformu, na které nám aplikace poběží. Já nechávám defaultní platformu „64bit Amazon Linux 2016.09“ a „Java 8 - v2.4.4“.

Důležitá je také záložka „**Security**“, ve které si po stisknutí tlačítka „Modify“ nastavíme klíč, kterým se do prostředí budeme přihlašovat. Potom co nám vyjede pravé vysouvací menu, vybereme v položce „EC2 key pair“ přislušný klíč pro přihlášení (vytvoření klíče je popsáno v kapitole 6.3.3). Poté stisknutím tlačítkem „Create Enviroment“ vytvoříme nové prostředí pro aplikaci. Vytvoření prostředí bude nějakou chvíli trvat.



Obrázek 41 AWS nově vytvořené prostředí aplikace

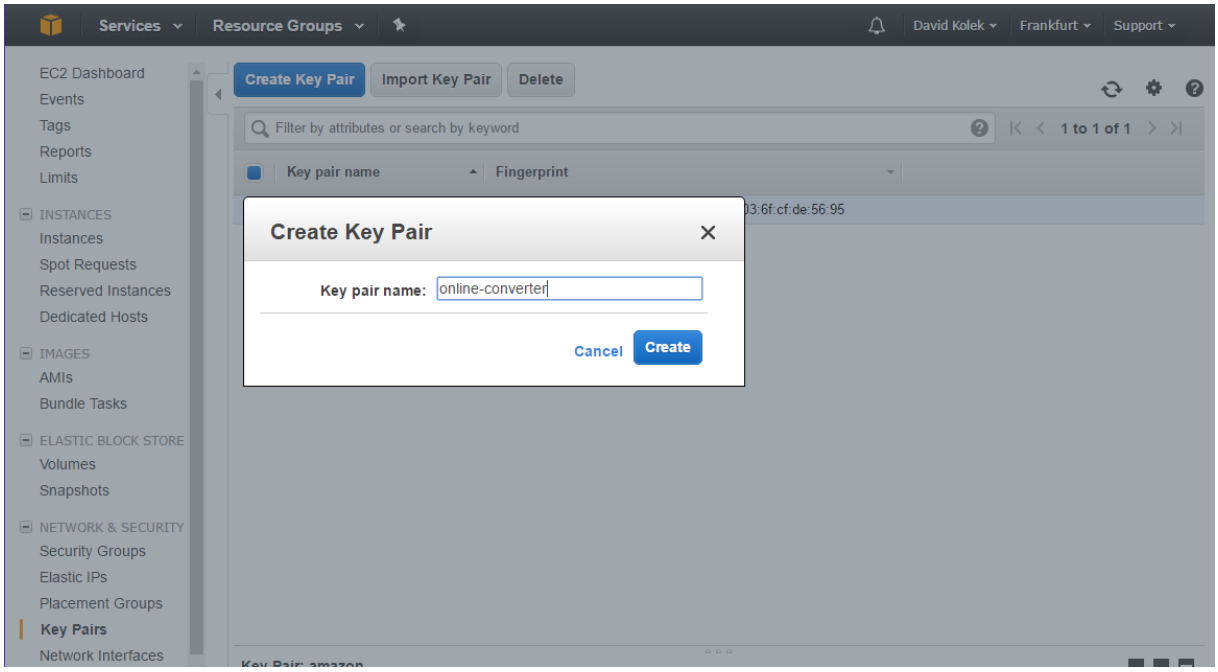
Po úspěšném vytvoření prostředí se nám na dříve zmíněné adrese spustí webový výstup naší aplikace. Defaultní zobrazení nově vytvořené aplikace můžete vidět na obrázku výše.

6.3.3 Konfigurace prostředí a instalace nezbytných součástí

Po úspěšném založení prostředí si do něj musíme vložit potřebné nástroje, které naše aplikace vyžaduje ke svému chodu. Vzhledem k tomu, že Java je již na serveru nainstalována, nemusíme se jí zabývat. Potřebujeme ale doinstalovat webový server Apache Tomcat ve správné verzi. Pro možnost konfigurace se musíme na server nejprve přihlásit, to můžeme provést prostřednictvím SSH. K tomu si však musíme vytvořit klíč, kterým se budeme přihlašovat.

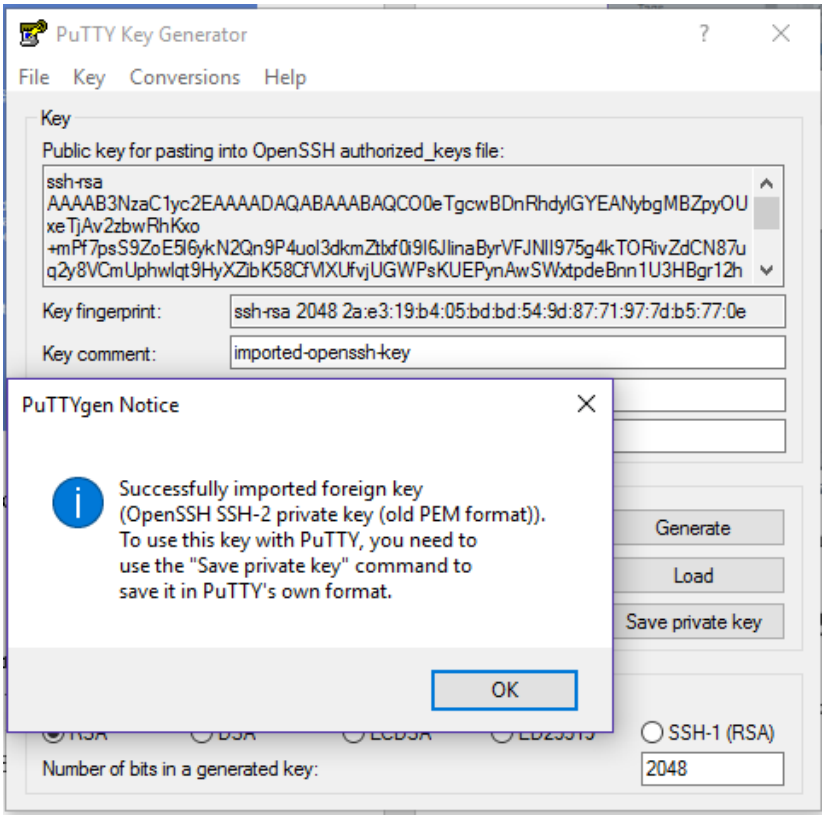
Pro vytvoření klíče nejprve vstoupíme zpět na úvodní portál AWS, kde si v menu „Services“ vybereme záložku „EC2“ (obrázek 37). Po přechodu na stránku této služby si v jejím levém posuvném menu najdeme položku „Key Pairs“ a klikneme na ní.

Na zobrazené stránce si pak klikneme na tlačítko „Create Key Pair“, které je umístěno v horní části stránky. Poté nám vyskočí okno, do toho vyplníme název klíče a klikneme na tlačítko „Create“ (obrázek 42). Tím se nám vytvoří klíč potřebný pro přihlášení na server. Okamžitě po stisknutí tlačítka „Create“ započne stahování klíče ve formátu „nasNazevKlice.pem“. PEM klíč přímo využijeme při přihlášení v Linuxu pomocí SSH. Pokud bychom se chtěli přihlásit z operačního systému Windows, musíme použít aplikaci Putty (pro přihlášení) a PuttyGen (pro převedení klíče na formát čitelný aplikací Putty). Obě aplikace se nám nainstalují při instalaci programu Putty.



Obrázek 42 Vytvoření Key Pair

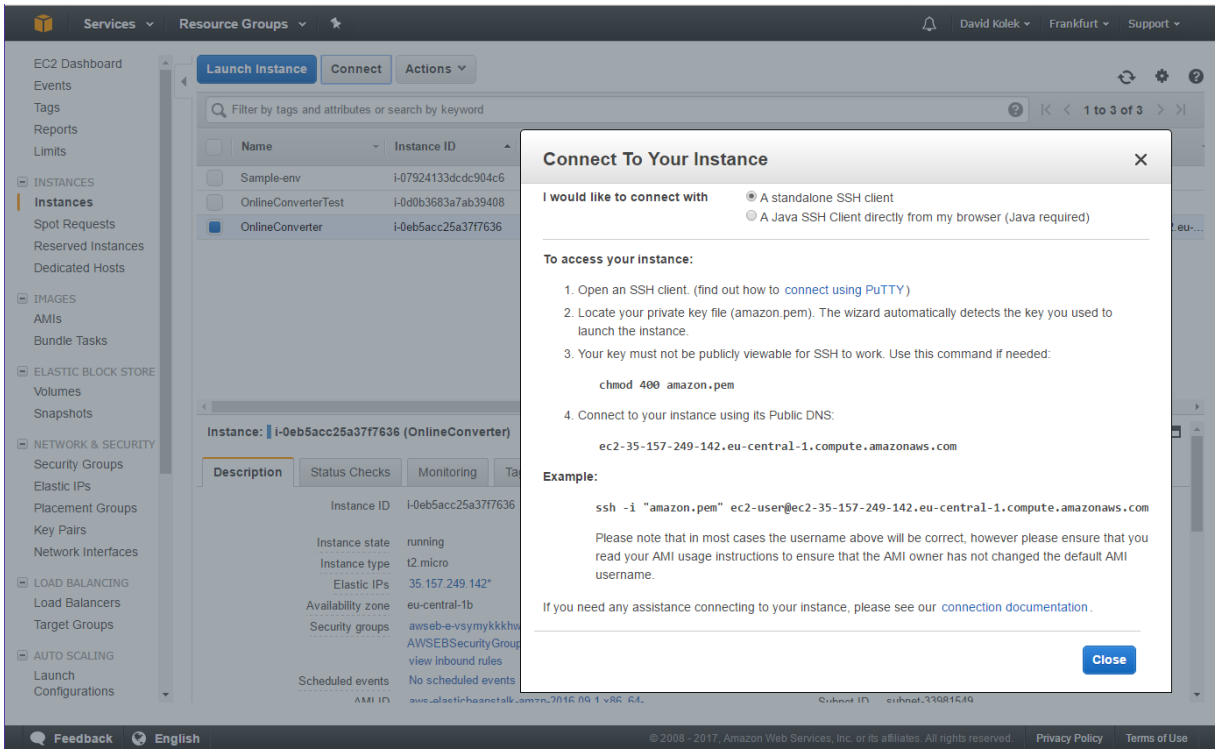
Nejprve si spustíme aplikaci PuttyGen a v záložce „File“ vybereme možnost „Load private key“ vyskočí nám okno s výběrem umístění. Najdeme v AWS vygenerovaný klíč, který byl stažen v předchozím kroku, a potvrdíme výběr. Poté se nám klíč načte do aplikace a vyskočí nám okno s upozorněním, že se jedná o formát PEM, který musíme pro použití s aplikací Putty převést na formát PPK. Stiskneme tlačítko „OK“ a poté tlačítkem „Save private key“ uložíme klíč již v podporovaném formátu PPK.



Obrázek 43 Převod PEM klíče na formát PPK

Vytvořený klíč si uchovejte pro opakované přihlášení na server, na nějakém bezpečném místě. Z AWS jej totiž nelze stáhnout opakovaně, můžete pouze vytvořit nový s jiným názvem.

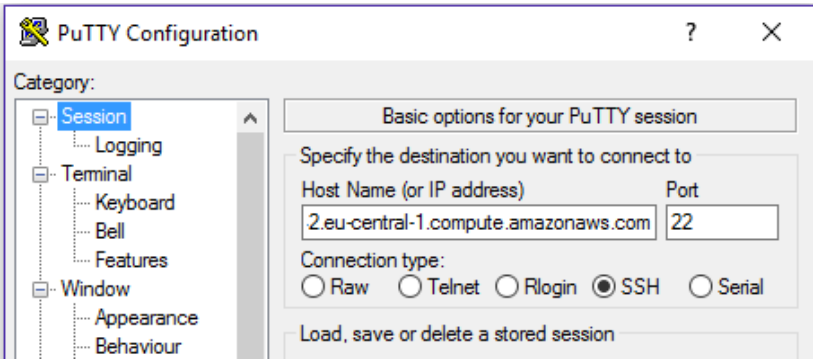
S tímto vytvořeným klíčem se již můžeme pomocí aplikace Putty přihlásit na server. Ještě však potřebujeme znát IP adresu a uživatelské jméno pro přihlášení. Tyto údaje nalezneme v umístění služby EC2, konkrétně v položce „Instances“ levého posuvného menu. Po kliknutí na tuto položku se nám zobrazí seznam všech vytvořených instancí v našem AWS účtu. Vybereme tu instanci, ke které se chceme připojit a klikneme na tlačítko „Connect“ v horní části stránky.



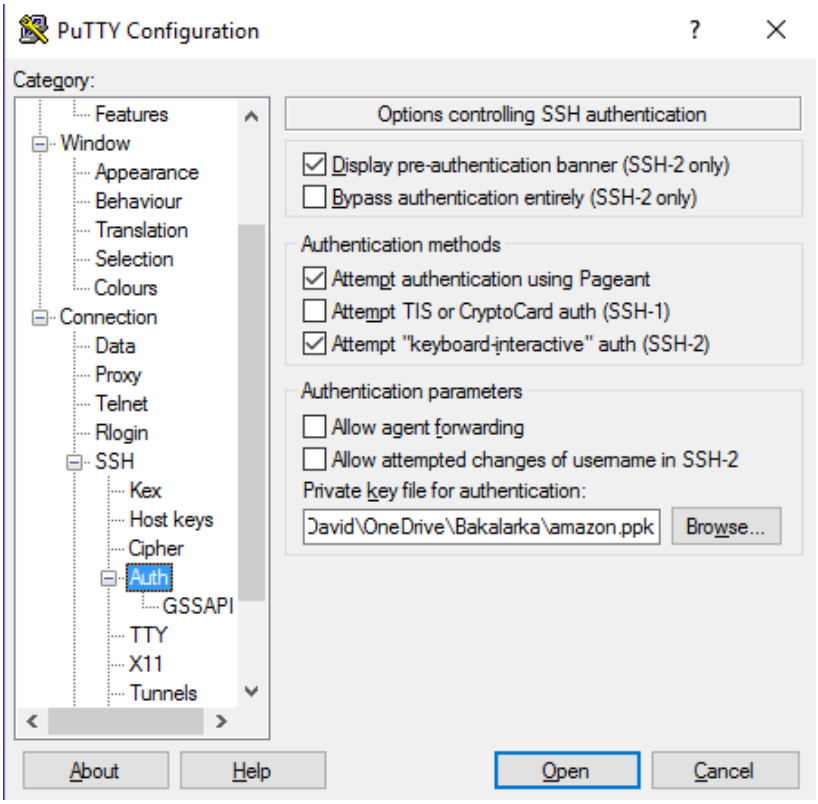
Obrázek 44 AWS zobrazení přihlašovacích údajů na server

Jak můžeme vidět na výše uvedeném obrázku, zobrazí se nám vyskakovací okno, kde je vepsána potřebná adresa „ec2-35-157-249-142.eu-central-1.compute.amazonaws.com“ a přihlašovací jméno „ec2-user“. Je zde uveden i SSH příkaz pomocí kterého se přihlásíme k serveru v systému Linux.

Tuto webovou adresu vložíme do aplikace Putty, do pole „Host name(or IP address)“ (obrázek 45). V záložce /SSH/Auth vybereme cestu k vytvořenému PPK klíči (obrázek 46) a klikneme na tlačítko „Open“ ve spodní části aplikace.

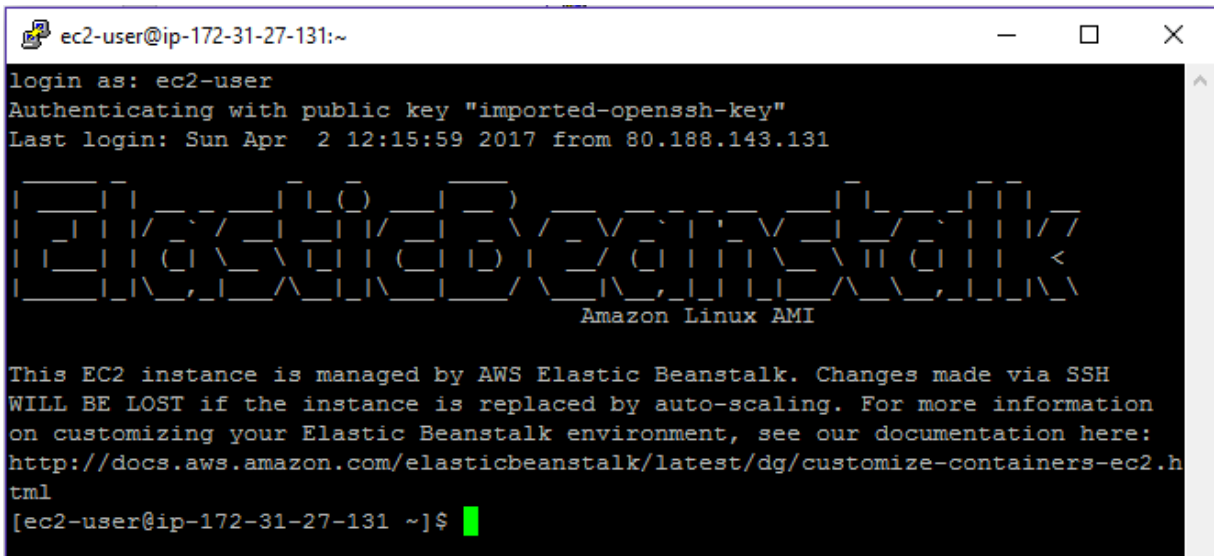


Obrázek 45 Připojení k AWS instanci pomocí Putty



Obrázek 46 Připojení k AWS instanci pomocí Putty a klíče PPK

Poté se nám otevře příkazová konzole, do které musíme pro dokončení přihlášení vepsat uživatelské jméno. Poté už můžeme pracovat přímo na námi vytvořené instanci serveru.



Obrázek 47 AWS úspěšné přihlášení na server

První věcí, kterou musíme doinstalovat, jej již zmíněný webový server Apache Tomcat. To provedeme pomocí příkazu:

```
sudo yum install tomcat8-webapps tomcat8-docs-webapp tomcat8-admin-webapps
```

Ten zajistí instalaci všech potřebných součástí Apache Tomcat (verze 8) na instanci našeho serveru. Defaultní port je 8080, takže po instalaci se nám aplikace rozjede na tomto portu našeho serveru. V mém

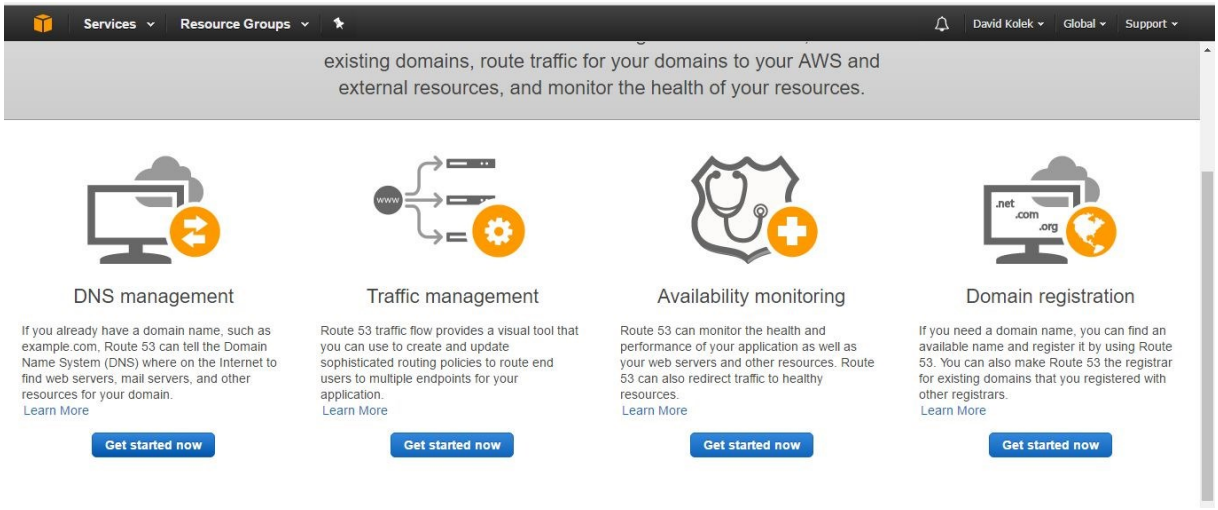
případě http://online-converter.eu-central-1.elasticbeanstalk.com:8080/. Nastavení aplikace, nasazení vytvořené webové aplikace a uživatelské účty jsou vysvětleny v předchozí kapitole 6.1 Apache Tomcat.

6.3.4 Registrace a nastavení DNS

Aby byl přístup k aplikaci snazší a pohodlnější, je potřeba vytvořenou instanci serveru s dlouhým názvem zamaskovat za kratší doménové jméno. Proto je nutné zaregistrovat název domény, který zatím nebyl zabrán příslušnou organizací. O české domény se stará sdružení cz.nic, kde můžete najít seznam registrátorů české domény https://www.nic.cz/whois/registrars/. Za českou doménu se však platí roční poplatek, což jsem nechtěl a vzhledem k tomu, že nepotřebuji českou doménu, hledal jsem registrátora bezplatných domén. Dopátral jsem se k webu www.freenom.com, kde si po registraci můžete zaregistrovat doménu zdarma na 1 rok. Já jsem si vybral „**onlineconverter.ga**“.

Po registraci této domény jsem jí ještě musel připojit k mé aplikaci na AWS. Upozorňuji, že **zavedení DNS do AWS není bezplatné**, pro každou nové doménové jméno je potřeba vytvořit tzv. „Hosted Zone“, které se v AWS zpoplatňuje 0,5 USD (bez daně) za prvních 25 zón.

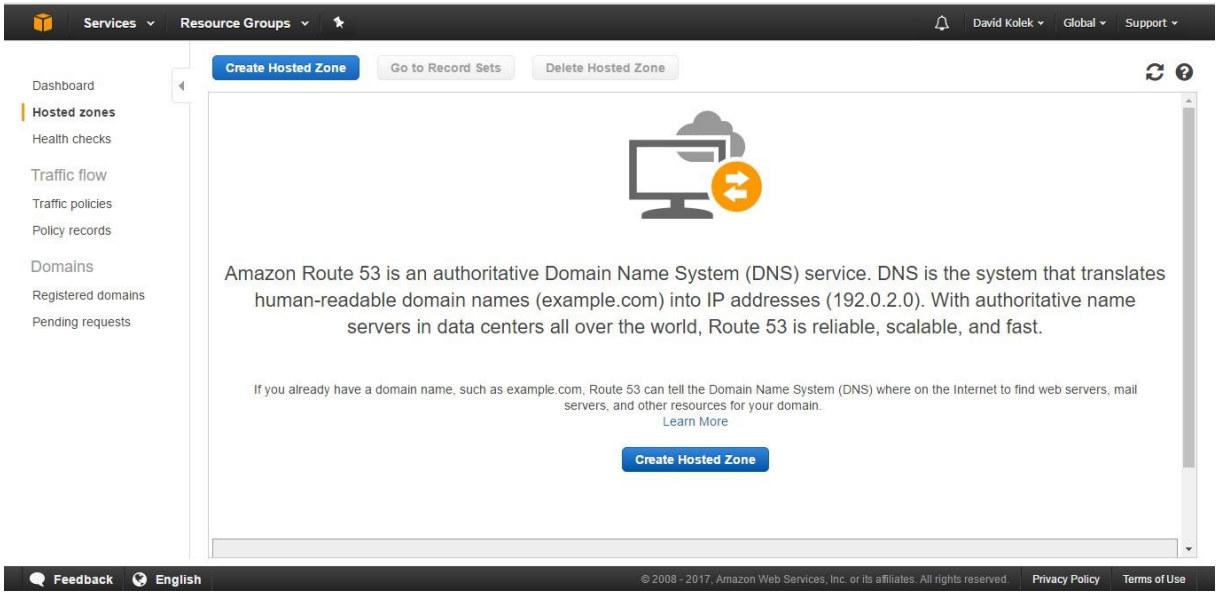
Pro registraci domény je třeba v záložce „Services“ (obrázek 37) kliknout na položku „Route 53“. Poté se objevíte v menu této služby, kde stisknete tlačítko „Get Started“ u položky „DNS management“ (níže uvedený obrázek).



Obrázek 48 AWS přidání domény (první spuštění)

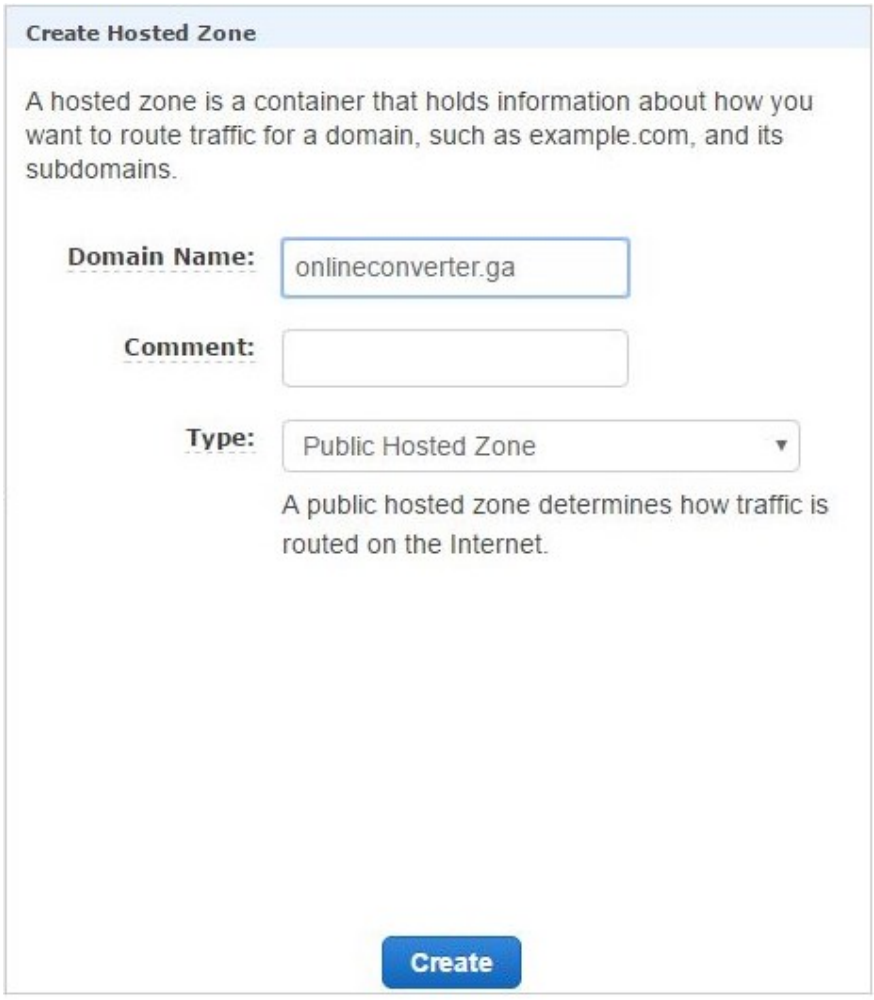
Výše uvedený obrázek představuje menu při prvním spuštěním položky „Route 53“, při opětovném spuštění bude stránka vypadat jinak, funkce se objevují v levém menu.

Dalším krokem je vytvoření takzvané „Hosted Zone“, Tu vytvoříme kliknutím na položku „Hosted zones“ v levém posuvném menu a poté kliknutím na tlačítko „Create Hosted Zone“ viz obrázek 49. Na obrázku je uvedeno první spuštění stránky, při opětovném spuštění se vzhled stránky lehce změní.

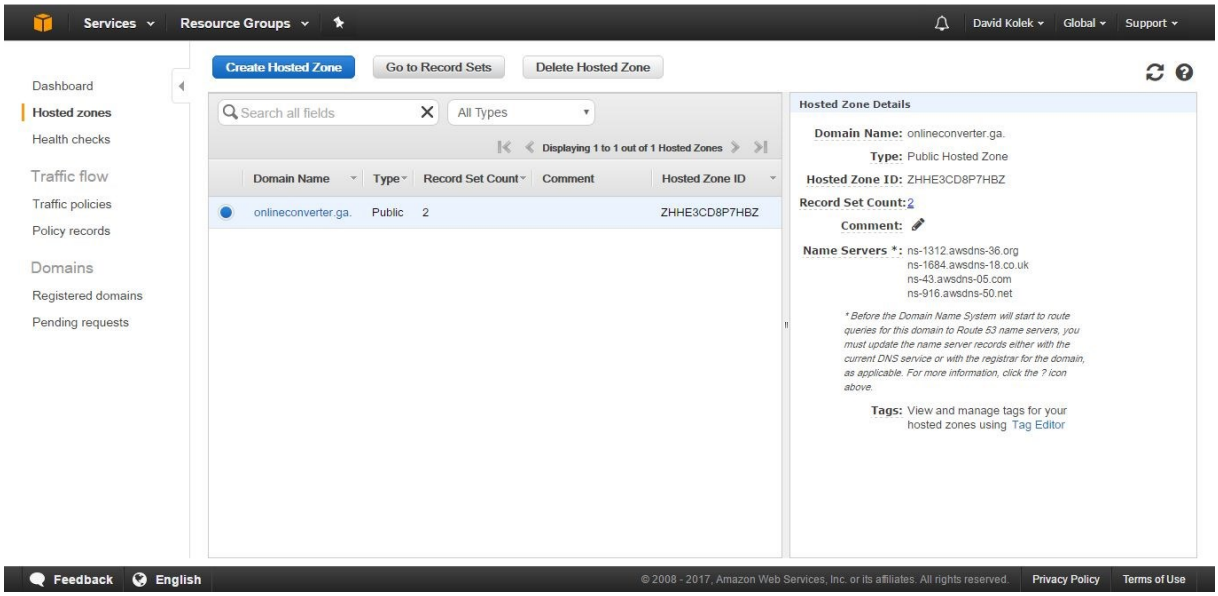


Obrázek 49 AWS vytvoření Hosted Zone

Po kliknutí na toto tlačítko nám vyjede v pravé části obrazovky panel, do kterého vyplníme název domény a klikneme na tlačítko „Create“ (obrázek níže).

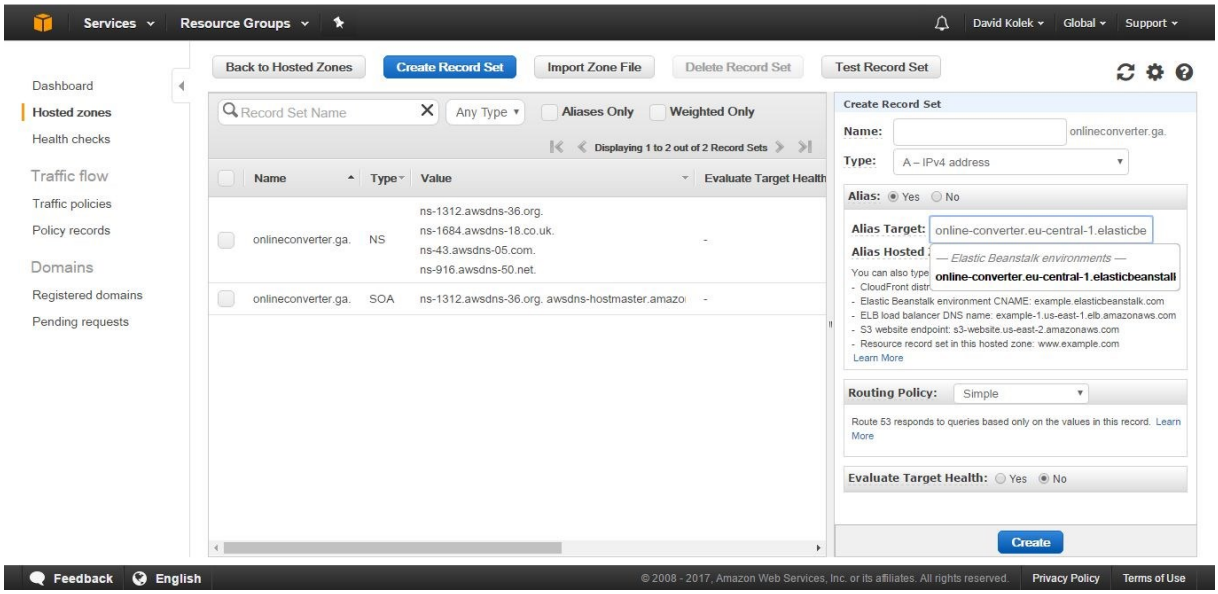


Obrázek 50 AWS přidání Hosted Zone - základní údaje



Obrázek 51 AWS úspěšné přidání Hosted Zone

Po úspěšném vložení Hosted Zone, se nám objeví v seznamu registrovaných zón. Když na ni klikneme, vypíší se nám její specifikace (pravý panel). Nejdůležitější z nich je informace o „Name Servers“ s kterými budeme ještě pracovat. Dále přejdeme kliknutím na název právě vytvořené zóny, nebo tlačítkem „Go to Record Set“.

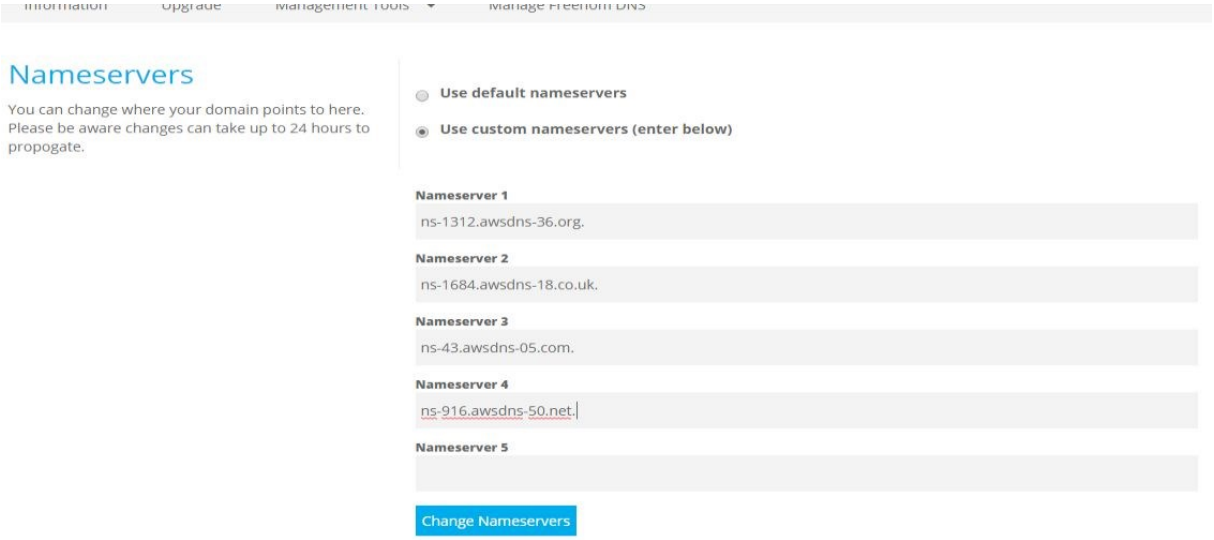


Obrázek 52 AWS vytvoření položky Record Set

Nyní musíme vytvořit tzv „Record Set“, který spojí instanci naší aplikace s vytvořenou zónou. Vytvoříme ji kliknutím na tlačítko „Create Record Set“. Vyjede nám pravý panel, do kterého vložíme potřebné údaje. Jméno necháme prázdné, typ necháme „A – IPv4 address“, pro alias zvolíme možnost „Yes“. Jako cíl aliasu zvolíme instanci našeho serveru, v mém případě „.online-converter.eu-central-1.elasticbeanstalk.com“ (po kliknutí vyjede výběrové menu). „Routing Policy“ zvolíme „Simple“ a poslední položku necháme na hodnotě „No“. Stiskneme tlačítko „Create“.

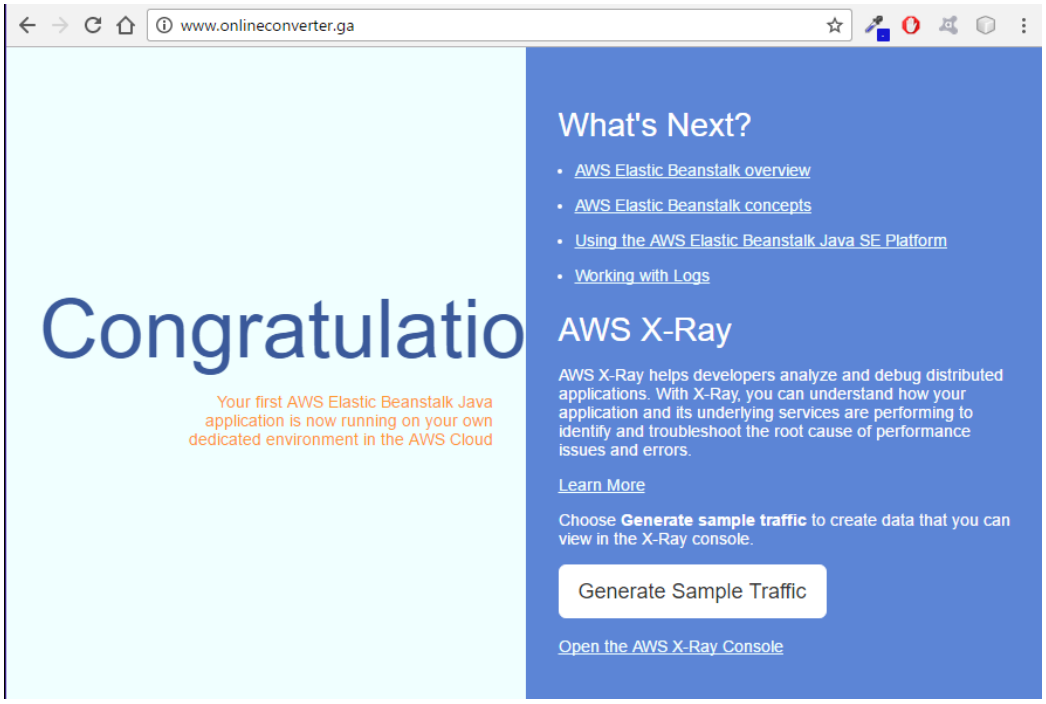
Nyní máme vytvořenou první Record Set, stejným postupem vytvoříme ještě druhý, který bude téměř stejný, rozdíl bude pouze v položce „Alias Target“, kde nyní vložíme instanci serveru s předponou „www“, v mém případě www.online-converter.eu-central-1.elasticbeanstalk.com.

V posledním kroku musíme zkopírovat jmenné servery (Name Servers), které můžeme vidět na obrázku 51. Tyto čtyři adresy vložíme do příslušných políček u námi zvoleného registrátora domény. Já je tedy vkládám k registrátorovi Freenom viz obrázek níže. Dejte si pozor, abyste nevložili adresu jmenného serveru s tečkou na konci, tu je potřeba z adresy při uložení nastavení u registrátora smazat.



Obrázek 53 Registrace NameServers u registrátora domény

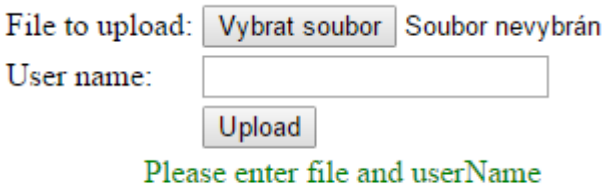
Nyní máme zaregistrovanou doménu k naší instanci serveru na AWS



Obrázek 54 AWS instance serveru s registrovanou doménou

6.4 Postup online aplikace

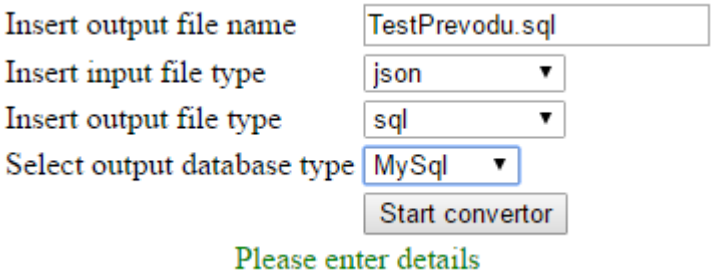
Pro převod souboru ze vstupu na výstup musí uživatel projít třemi kroky mé aplikace. Prvním krokem je vložení vstupního souboru a uživatelského jména, pod kterým bude aplikace uložena (postup pro anonymní uživatele).



Obrázek 55 Postup webovou aplikací – krok 1

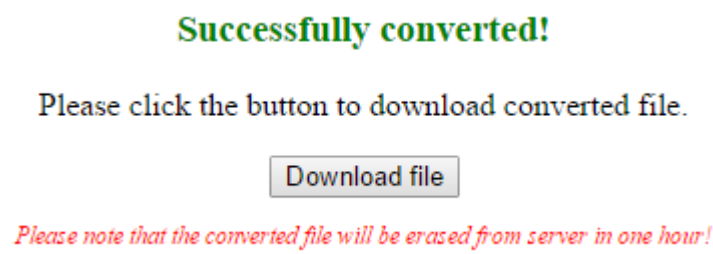
Po kliknutí na tlačítko „Upload“ se v případě korektně zadaných údajů začne nahrávat soubor od uživatele na server, kde bude později zpracován. Nahrání souboru může trvat i delší dobu v závislosti na jeho velikosti a datovém připojení uživatele.

Po úspěšném nahrání souboru vyskočí uživateli webová stránka, do které je třeba zadat základní údaje potřebné ke spuštění a dokončení převodu.



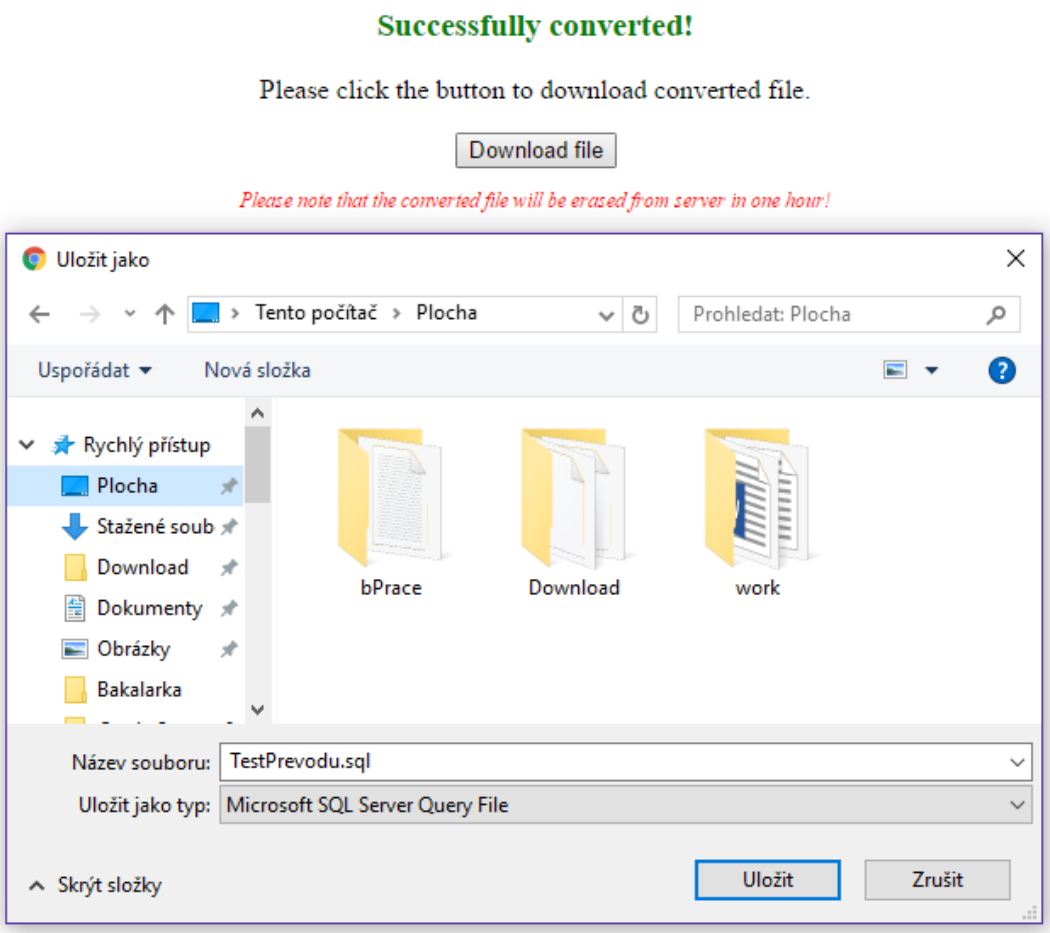
Obrázek 56 Postup webovou aplikací – krok 2

Zde musí uživatel zadat název pro výstupní soubor, a to buď s příslušnou příponou, nebo bez ní. Musí vybrat typ vstupního a výstupního formátu a v případě výstupu typu SQL musí uživatel zadat i typ databáze, do které chce soubor převést. Po zadání všech údajů už může uživatel stisknout tlačítko „Start convertor“, které spustí konverzi na výstupní formát.



Obrázek 57 Postup webovou aplikací – krok 3

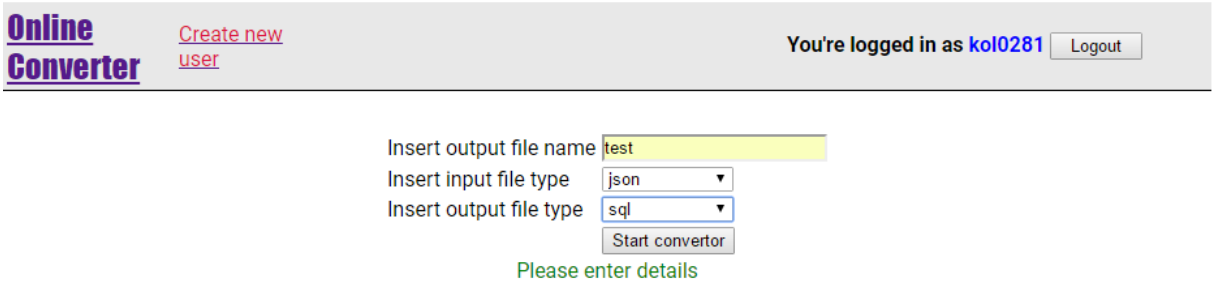
Po dokončení převodu, jehož délka je závislá na typu, složitosti a velikosti vstupního souboru, bude uživatel přesměrován na webovou stránku, která mu umožní stáhnout převedený soubor (obrázek 57).



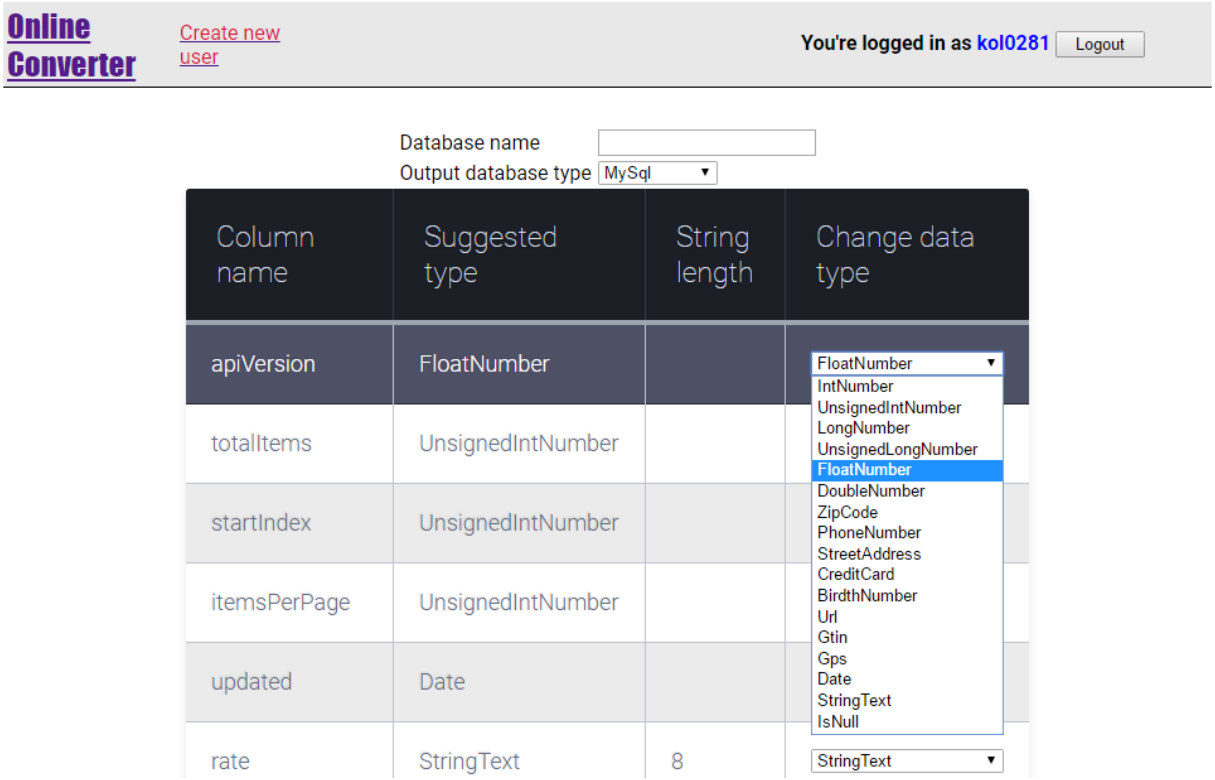
Obrázek 58 Postup webovou aplikací – krok 3, uložení souboru

Stáhnutí souboru uživatel provede stisknutím tlačítka „Download file“. Okamžitě po jeho zmáčknutí, se uživateli začne stahovat soubor do jeho zařízení. Ukázka této akce je zobrazena na obrázku 58.

Níže uvedený obrázek reprezentuje stránku, která je zobrazena přihlášenému uživateli. V tomto příkladu se jedná o uživatele „kol0281“ s administrátorskými právy, takže má možnost vytvořit uživatele po kliknutí na odkaz „Create new user“.



Obrázek 59 Přihlášený uživatel – Admin



Obrázek 60 Konfigurace SQL parseru

V případě, kdy přihlášený uživatel vybere na Obrázek 59, jako výstupní formát SQL a klikne na tlačítko „Start converter“, zobrazí se mu stránka, na které se vypíše provedená analýza vstupního souboru. Uživatel si zde bude moct konfigurovat typy jednotlivý sloupců výstupní databáze. Může si zde zvolit i název databáze a její typ. Po potvrzení nastavení se uživatel dostane na stránku pro stažení výstupního souboru (Obrázek 58).

6.5 Testování aplikace a testovací soubory

Pro testování aplikace převodníku, jsem vyhledal hned několik korektně vytvořených souborů, různých datových typů. Mezi testované formáty jsem zahrnul všechny podporované formáty mé aplikace. Testoval jsem jak obecné soubory, tak soubory používané v praxi (XML výpis produktů eshopu). Pro testování funkčnosti jednotlivých převodníků jsem zprvu využíval malých testovacích souborů, s pouze omezeným množstvím dat. Po konečné implementaci jsem však přešel na testování větších datových souborů. Největší testovaný soubor měl velikost cca. 52 MB a jednalo se o soubor s exportovanými informace eshopu. Původní formát tohoto souboru byl XML a převedl jsem jej na všechny podporované formáty. Tento soubor a všechny formáty, do kterého byl převeden naleznete ve složce „**Přílohy/Testovací Soubory**“. V této složce se nacházejí i další testovací soubory.

Jak již jsem psal v předešlých kapitolách, aplikace parseru je silně závislá na operační paměti. Proto je na mé nasazené online aplikaci možno zpracovat pouze datové soubory menší velikosti. Potřeba paměti je přímo závislá na velikosti a případnému stupni zanoření konkrétního formátu. Online aplikaci jsem z toho důvodu nastavil na příjem souboru o maximální velikosti 3 MB. Pro testování lze využit soubory v příloze.

Aplikace byla nasazena na adresu http://onlineconverter.ga/, kde si uživatel může spustit parser a to jako anonymní uživatel, který nemá možnost nastavení datových typů při výstupu SQL. Pro možnost zobrazení a nastavení analýzy musí být uživatel přihlášen, a to třeba jako uživatel „**use1**“ pod heslem „**pass**“. Registraci nového uživatele může provést pouze uživatel s administrátorskými právy.

Veškeré soubory se zdrojovými kódy jsou obsaženy v příloze ve složce „Zdrojové Kódy/Converter“ pro zdrojové kódy aplikace parseru a „Zdrojové Kódy/Webapp“ pro webovou aplikaci. Ostatní soubory jsou uvedeny ve složce „ostatní“

7 Závěr

V závěru této práce bych chtěl zmínit skutečnost, že téma této bakalářské práce je nadčasové, vzhledem k tomu, že se stále objevují nové a nové formáty pro ukládání dat. Tato práce se zabývá pouze formáty, které jsou často používané jak pro soukromé účely, tak pro firemní využití. Vzhledem k těmto faktům lze uvést, že toto řešení problému se dá vždy inovovat a přidat další vstupní / výstupní datové formáty, bez potřeby změny ostatních vstupních, výstupních nebo analytických funkcí mé aplikace.

Při implementaci této práce jsem se setkal z řadou problémů, kterým jsem musel čelit a vyřešit. Mezi první problémy patřila složitost některých datových struktur (json, xml), které jsem do té doby neznal. Nejvíce problematickou částí této práce byl právě průchod formátem JSON, který jsem musel procházet tak, abych z něj dostal všechny potřebné informace co nejefektivněji. Toto se mi podle mého názoru podařilo.

Mezi další problematiku, kterou se mi podařilo úspěšně vyřešit, bylo samotné importování externích knihoven do mého programu, případně vytvoření takové knihovny (souboru JAR) v případě, kdy jsem potřeboval spojit aplikaci parteru a webovou aplikaci. Takové problémy jsme v běžné výuce mého studia vůbec neřešili, takže je bylo potřeba nastudovat a vyřešit právě až při implementaci této práce.

Vzhledem k tomu, že jsem neměl předmět zabývající se vytvářením webového rozhraní v jazyce Java, i tento problém jsem musel řešit sám. Zde jsem si musel nastudovat, jak se takové rozhraní vůbec vytváří, a co vše je k tomu potřeba. Díky tomu jsem se naučil pracovat s webovým serverem Apache Tomcat a případně ho i upravit podle svých potřeb. Přínosná byla i práce s frameworkem Spring MVC, kde jsem řešil „banální“ problematiku vytváření cesty v URL adrese pomocí anotací. Tento problém jsem řešil několik dní, protože mi tyto cesty, ačkoli by měly, vůbec nefungovaly. Problém jsem nakonec vyřešil banálním způsobem, a to byla instalace novější verze programovacího prostředí Netbeans, chyba totiž evidentně spočívat právě v jeho nastavení.

V neposlední řadě jsem musel řešit problém nasazení mé aplikace na web tak, aby byla přístupná online, což byla jedna z dalších věcí, která nebyla součástí výuky mého bakalářského studia. To jsem provedl pomocí již zmiňovaného AWS, o kterém jsem do této doby nic nevěděl. Při nasazování aplikace jsem se naučil využívat některé funkce těchto služeb Amazonu. Příkladem je vytvoření serveru, jeho ovládání, nebo registrace DNS pro mou aplikaci. Při nasazení na servery Amazonu jsem narazil také na problém kompatibility, který ač neměl nastat, nastal. Ač jsem na server nainstaloval stejné verze aplikací, které můj program využívá k chodu, tak jeho funkčnost byla jiná než při spuštění na mém zařízení.

Jak již jsem zmínil dříve, mnou vytvořenou aplikaci lze v budoucnu rozšířit o další vstupní nebo výstupní formáty podle aktuální potřeby. Lze taky doplnit nebo vylepšit použité analyzační funkce mé aplikace a využít je pro specifičtější datový výstup. Například lze dopracovat myšlenku cizích klíčů, při výstupu typu SQL, kde má aplikace sice zjistí jednotlivé cizí klíče k daným tabulkám, ale již není implementováno jejich zakomponování do výstupního souboru.

Oproti ostatní podobným aplikacím, ať už online nebo desktopových, nabízí má aplikace možnost nastavení datových typů při výstupu SQL. Nevýhodou mé aplikace je jednoznačně malá velikost vstupního souboru, která je zapříčiněna právě omezenými zdroji bezplatného serveru.

8 Literatura

[1] The Comma Separated Value (CSV) File Format. Creativyst Software [online]. 2010 [cit. 2017-03-14]. Dostupné z: http://creativyst.com/Doc/Articles/CSV/CSV01.htm

[2] Microsoft Compound Document File Format [online]. 2007 [cit. 2017-03-14]. Dostupné z: http://www.openoffice.org/sc/compdocfileformat.pdf

[3] Microsoft Excel File Format [online]. 2008 [cit. 2017-03-14]. Dostupné z: https://www.openoffice.org/sc/excelfileformat.pdf

[4] Introducing the Office (2007) Open XML File Formats [online]. 2006 [cit. 2017-03-14]. Dostupné z: https://msdn.microsoft.com/en-us/library/aa338205%28v=office.12%29.aspx?f=255&MSPPError=-2147217396

[5] XML ESSENTIALS [online]. 2015 [cit. 2017-03-14]. Dostupné z: https://www.w3.org/standards/xml/core

[6] Extensible Markup Language (XML) 1.0 (Fifth Edition) [online]. 2008 [cit. 2017-03-14]. Dostupné z: https://www.w3.org/TR/REC-xml/

[7] KOSEK, Jiří. Základy jazyka XML [online]. [cit. 2017-03-14]. Dostupné z: http://www.kosek.cz/clanky/swn-xml/syntaxe.html

[8] HAVEL, Jakub. XML pro web aneb od teorie k praxi - 2 díl [online]. [cit. 2017-03-15]. Dostupné z: http://www.zive.cz/clanky/xml-pro-web-aneb-od-teorie-k-praxi-2dil/sc-3-a-109709/default.aspx

[9] HAVEL, Jakub. XML pro web aneb od teorie k praxi - 3 díl [online]. [cit. 2017-03-15]. Dostupné z: http://www.zive.cz/clanky/xml-pro-web-aneb-od-teorie-k-praxi---3dil/sc-3-a-109776/default.aspx

[10] KOSEK, Jiří. DTD a automatická kontrola struktury dokumentu [online]. [cit. 2017-03-15]. Dostupné z: http://www.kosek.cz/clanky/swn-xml/dtd.html

[11] KOSEK, Jiří. Deklarace atributů [online]. [cit. 2017-03-15]. Dostupné z: http://www.kosek.cz/clanky/swn-xml/ar04s48.html

[12] HAVEL, Jakub. XML pro web aneb od teorie k praxi - 5.díl [online]. 2003 [cit. 2017-03-15]. Dostupné z: http://www.zive.cz/clanky/xml-pro-web-aneb-od-teorie-k-praxi---5dil/sc-3-a-109862/default.aspx

[13] KOSEK, Jiří. Připojení DTD k dokumentu [online]. [cit. 2017-03-15]. Dostupné z: http://www.kosek.cz/clanky/swn-xml/ar04s49.html

[14] An Introduction to StAX [online]. Elliotte Rusty Harold, 2003 [cit. 2017-03-15]. Dostupné z: http://www.xml.com/pub/a/2003/09/17/stax.html

[15] Úvod do JSON [online]. [cit. 2017-03-15]. Dostupné z: http://www.json.org/json-cz.html

[16] The JSON Data Interchange Format [online]. Ecma International, 2013 [cit. 2017-03-15]. Dostupné z: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

[17] BEN-KIKI, Oren, Clark EVANS a Ingy DÖT NET. YAML Ain't Markup Language (YAML™) Version 1.2 [online]. 2009 [cit. 2017-03-21]. Dostupné z: http://www.yaml.org/spec/1.2/spec.html

[18] MALÝ, Martin. YAML: Serializační formát pro ukládání dat [online]. 2009 [cit. 2017-03-21]. Dostupné z: https://www.zdrojak.cz/clanky/yaml-serializacni-format-pro-ukladani-dat/

[19] KRÁTKÝ, Michal a Radim BAČA. Databázové systémy. 17. listopadu 15, 708 33 Ostrava–Poruba Česká republika, 2015. VSB – Technická univerzita Ostrava.

[20] About MySQL. MySQL [online]. Oracle Corporation, 2017 [cit. 2017-04-10]. Dostupné z: https://www.mysql.com/about/

[21] CREATE TABLE Syntax. MySQL [online]. Oracle Corporation, 2017 [cit. 2017-04-10]. Dostupné z: https://dev.mysql.com/doc/refman/8.0/en/create-table.html
[22] Inserting Data into A Table Using MySQL INSERT Statement. MySQLTutorial [online]. [cit. 2017-04-11]. Dostupné z: http://www.mysqltutorial.org/mysql-insert-statement.aspx
[23] Oracle Database: Objevte špičkovou databázi Oracle Database v cloudu. ORACLE [online]. [cit. 2017-04-11]. Dostupné z: https://www.oracle.com/cz/database/index.html
[24] CONSTRAINT clause. ORACLE [online]. [cit. 2017-04-11]. Dostupné z: http://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqlj13590.html
[25] CONSTRAINT clause: INSERT ALL Statement. TechONTHENet [online]. [cit. 2017-04-11]. Dostupné z: https://www.techonthenet.com/oracle/questions/insert_rows.php
[26] Announcing SQL Server on Linux. Microsoft [online]. 2016 [cit. 2017-04-11]. Dostupné z: https://blogs.microsoft.com/blog/2016/03/07/announcing-sql-server-on-linux/#sm.0000cjc0zih80er4pma2c6wb2cdjb
[27] SQL Server: CREATE TABLE Statement. TechONTHENet [online]. [cit. 2017-04-11]. Dostupné z: https://www.techonthenet.com/sql_server/tables/create_table.php
[28] CREATE TABLE (Transact-SQL). Microsoft: Microsoft TechNet [online]. [cit. 2017-04-11]. Dostupné z: https://technet.microsoft.com/en-us/library/ms174979(v=sql.105).aspx
[29] PostgreSQL. PostgreSQL [online]. 2017 [cit. 2017-04-11]. Dostupné z: http://postgres.cz/wiki/PostgreSQL
[30] PostgreSQL 9.4.11 Documentation: Create Table. PostgreSQL [online]. [cit. 2017-04-11]. Dostupné z: https://www.postgresql.org/docs/9.4/static/sql-createtable.html
[31] PostgreSQL 9.4.11 Documentation: Insert. PostgreSQL [online]. [cit. 2017-04-11]. Dostupné z: https://www.postgresql.org/docs/9.4/static/sql-insert.html
[32] Neon sandbox. NEON sandbox [online]. Nette Foundation, 2017 [cit. 2017-04-11]. Dostupné z: https://ne-on.org/
[33] Apache Tomcat. Apache Tomcat [online]. The Apache Software Foundation, 2017 [cit. 2017-04-11]. Dostupné z: http://tomcat.apache.org/
[34] Web MVC framework: Introduction to Spring Web MVC framework. SpringbyPivotal [online]. Pivotal Software [cit. 2017-04-11]. Dostupné z: https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html
[35] Amazon EC2 Pricing. Amazon Web Services [online]. Amazon Web Services, 2017 [cit. 2017-04-11]. Dostupné z: https://aws.amazon.com/ec2/pricing/on-demand/
[36] About CodeBeautify. Code Beautify [online]. [cit. 2017-04-14]. Dostupné z: http://codebeautify.org/aboutus
[37] Convert CSV/Excel To JSON,XML,HTML,SQL.. Convert CSV [online]. [cit. 2017-04-14]. Dostupné z: http://www.convertcsv.com/
[38] FREEFORMATTER.COM. Free Online Tool For Developers - FreeFormatter.com [online]. [cit. 2017-04-14]. Dostupné z: http://www.freeformatter.com/
[39] Business Process Automation Software: Automate Inbound Message Processing. ThinkAutomation [online]. [cit. 2017-04-14]. Dostupné z: http://www.thinkautomation.com/
[40] Log Parser 2.2. Microsoft [online]. [cit. 2017-04-14]. Dostupné z: https://www.microsoft.com/en-us/download/details.aspx?id=24659
[41] LogQL: Log Parser [online]. c2017 [cit. 2017-04-14]. Dostupné z: http://www.logql.com/

[42] Data Parse Free [online]. National Data Parsing Canada Corporation, c1986-2017 [cit. 2017-04-14]. Dostupné z: https://www.dataparse.com/free.aspx

9 Přílohy